华为“智能基座”系列课程

《基于 MindSpore 的目标检测（YOLOv3）》

版本：2.2

华为技术有限公司

商标声明

和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://e.huawei.com

目录

1 YOLOv3 的目标检测实验

1.1 实验介绍

1.1.1 实验背景

目标检测是很多计算机视觉应用的基础，它结合了目标分类和定位两个任务。深度学习用于目标检测的算法从思路上来看，可以分为两大类，一类是 two stage 的方法，也就是把整个分为两部分，生成候选框和识别框内物体，例如 R-CNN 系列；另一类是 one stage 的方法，把整个流程统一在一起，直接给出检测结果，主要包含 SSD,YOLO 系列。目标检测的 backbone 一般是基于 ImageNet 预训练的图像分类网络。图像分类问题只关注分类和感受视野，不用关注物体定位，但是目标检测领域同时很关注空间信息。如果下采样过多，会导致最后的 feature map 很小，小目标很容易漏掉。很多基础架构网络，比如 ResNet、FPN 等神经网络提取图像的上下文信息，不断在特征提取方向优化。

本实验使用中园区摄像头采集到的现实数据集，采用 MindSpore 深度学习框架构建 YOLOv3 网络，在华为云平台的 ModelArts 上创建基于昇腾 910 处理器的训练环境，启动训练并得到目标检测的模型，识别进出人员是否佩戴口罩的功能。

1.1.2 实验目的

- 掌握 YOLOv3 模型的基本结构和编程方法。
- 掌握使用 YOLOv3 模型进行目标检测。

1.1.3 实验清单

实验	简述	难度	开发环境
YOLOv3 的目标检测实验	本实验使用中园区摄像头采集到的现实数据集，采用 MindSpore 深度学习框架构建 YOLOv3 网络，在华为云平台的 ModelArts 上创建基于昇腾 910 处理器的训练环境，启动训练并得到目标检测的模型，识别进出人员是否佩戴口罩的功能。	中级	ModelArts：MindSpore1.5，Python3.7， Ascend 910 +ARM

1.2 实验环境

1.2.1 实验环境介绍

- MindSpore 训练：在华为云 ModelArts 上的开发环境中执行。

实验环境	实验平台	AI 计算框架	AI 处理器/算力	软件
MindSpore 训练	华为云 ModelArts	MindSpore1.5	Ascend 910	Notebook 环境， Python3.7.5，MindSpore 1.5

表1.2.1.1.1.1.1.1 环境要求

实验平台介绍：

- 华为云 ModelArts 平台：https://www.huaweicloud.com/product/modelarts.html ModelArts 是面向开发者的一站式 AI 平台，为机器学习与深度学习提供海量数据预处理及交互式智能标注、大规模分布式训练、自动化模型生成，及端-边-云模型按需部署能力，帮助用户快速创建和部署模型，管理全周期 AI 工作流。

AI 计算框架介绍：

- MindSpore 深度学习框架：https://www.mindspore.cn/mindspore

昇思 MindSpore 是一个全场景深度学习框架，旨在实现易开发、高效执行、全场景覆盖三大目标，提供支持异构加速的张量可微编程能力，支持云、服务器、边和端多种硬件平台。

NPU 芯片介绍：

- Ascend 910：https://e.huawei.com/cn/products/cloud-computing-dc/atlas/ascend-910

昇腾 910 是一款具有超高算力的 AI 处理器，其最大功耗为 310W，华为自研的达芬奇架构大大提升了其能效比。八位整数精度（INT8）下的性能达到 640TOPS，16 位浮点数（FP16）下的性能达到 320 TFLOPS。

- Ascend 310：https://e.huawei.com/cn/products/cloud-computing-dc/atlas/ascend-310

昇腾 310 是一款高效、灵活、可编程的 AI 处理器。基于典型配置，八位整数精度（INT8）下的性能达到 22TOPS，16 位浮点数（FP16）下的性能达到 11 TFLOPS，而其功耗仅为 8W。昇腾 310 芯片采用华为自研的达芬奇架构，集成了丰富的计算单元，在各个领域得到广泛应用。随着全 AI 业务流程的加速，昇腾 310 芯片能够使智能系统的性能大幅提升，部署成本大幅降低。

1.2.2 实验环境准备

请参考下方实验手册《MindSpore 环境搭建实验手册》的“4 线上服务环境搭建”部分。

1.3 数据与模型

1.3.1 数据集介绍

目标检测的数据集一般需要标记出被检测目标的区域位置和类别。

本实验的数据集取自华为园区实验摄像头采集到的人员进出信息，并对画面中出现的人（person）、脸（face）、口罩（mask）进行标注，分别标注出位置坐标和标签类别，得到包含 500 张图像（jpg 格式）和标签（xml 格式）的训练集和 8 张图像（jpg 格式）的测试集。其中 xml 文件中记录了物体框的坐标和类别等信息。

数据集下载链接： https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/deep-learning/data.zip

*说明：data 数据集大小 91.3 MB，实验代码中可直接下载，无需上传。

data 数据集目录如下：

```
├─data
├─train    # 训练数据集.
├─jpg      # 训练集图片
├─xml      # 训练集标签
├─test     # 测试数据集
├─jpg      # 测试集图片
```

jpg 为图像数据，图像尺寸不固定，多为 1920*1080，部分为 1280*720。

xml 为标签数据，与图片一一对应，包含了物体框的坐标和类别等信息。

图片示例：

图1.3.1.1.1.1.1 数据集图片

标签示例：

每个图片对应的 xml 文件中都包含了该图片的基本信息，比如文件名、来源、图像尺寸以及图像中包含的物体区域信息和类别信息等。

xml 文件具体字段如下：

- folder：文件夹；
- filename：图像名称；

- path：路径；
- source：来源；
- size：图像尺寸，包含图像宽（width）、高（height）、深度（depth）；
- object：物体区域信息和类别信息， 同张图片可有多个 object 字段；

object 字段具体内容如下：

标签	说明
name	物体类别名称，本实验共有三个类别，分别是“person”、“face”、“mask”；
pose	目标物体姿态描述
truncated	被截断，物体的遮挡超过 15-20%并且位于边界框外
difficult	难以识别的物体
bndbox	（xmin, ymin）左下角坐标，（xmax, ymax）右上角坐标

表1.3.1.1.1.1.1.1 Object 字段内容

图 1 对应的 xml 文件内容：

```
<annotation>
    <folder>02-03_video01_frames</folder>
    <filename>02-02_video12-frame-00249.jpg</filename>
    <path>D:\EI Projects\1 项目交流\园区 未戴口罩识别\戴口罩监控视频\02-03\02-
03_video01_frames\02-02_video12-frame-00249.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>1920</width>
        <height>1080</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>person</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>335</xmin>
            <ymin>402</ymin>
            <xmax>587</xmax>
            <ymax>615</ymax>
        </bndbox>
    </object>
    <object>
```

```
        <name>person</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>736</xmin>
            <ymin>345</ymin>
            <xmax>970</xmax>
            <ymax>515</ymax>
        </bndbox>
    </object>
    <object>
        <name>person</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>255</xmin>
            <ymin>762</ymin>
            <xmax>590</xmax>
            <ymax>1037</ymax>
        </bndbox>
    </object>
    <object>
        <name>face</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>816</xmin>
            <ymin>383</ymin>
            <xmax>884</xmax>
            <ymax>471</ymax>
        </bndbox>
    </object>
    <object>
        <name>face</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>391</xmin>
            <ymin>429</ymin>
            <xmax>465</xmax>
            <ymax>545</ymax>
        </bndbox>
    </object>
```

```
<object>
    <name>face</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
        <xmin>327</xmin>
        <ymin>814</ymin>
        <xmax>443</xmax>
        <ymax>966</ymax>
    </bndbox>
</object>
<object>
    <name>mask</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
        <xmin>392</xmin>
        <ymin>489</ymin>
        <xmax>461</xmax>
        <ymax>542</ymax>
    </bndbox>
</object>
<object>
    <name>mask</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
        <xmin>823</xmin>
        <ymin>429</ymin>
        <xmax>884</xmax>
        <ymax>468</ymax>
    </bndbox>
</object>
<object>
    <name>mask</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
        <xmin>329</xmin>
        <ymin>892</ymin>
        <xmax>436</xmax>
        <ymax>964</ymax>
    </bndbox>
```

```
    </object>
</annotation>
```

1.3.2 模型介绍

- YOLO 网络介绍

YOLO 是单阶段方法的开山之作。它将检测任务表述成一个统一的、端到端的回归问题，并且以只处理一次图片同时得到位置和分类而得名。

YOLOV1 是典型的目标检测 one stage 方法，用回归的方法去做目标检测，执行速度快，达到非常高效的检测。YOLOV1 的基本思想是把一副图片，首先 reshape 成 448×448 大小（由于网络中使用了全连接层，所以图片的尺寸需固定大小输入到 CNN 中），然后将其划分成 SxS 个单元格（原文中 S=7），如果目标中心点在某个单元格内，该单元格就负责预测该目标。输出层的大小为 7x7，通道数为 30。7x7 可以看作将原图分为 7x7 的网格，而每个格子中有 30 个数。这三十个数分别对应了两组（意味着每个网格尝试着预测两个边界框）的“位置信息+置信度”以及 20 个类别（VOC 数据集中有 20 个类别）。

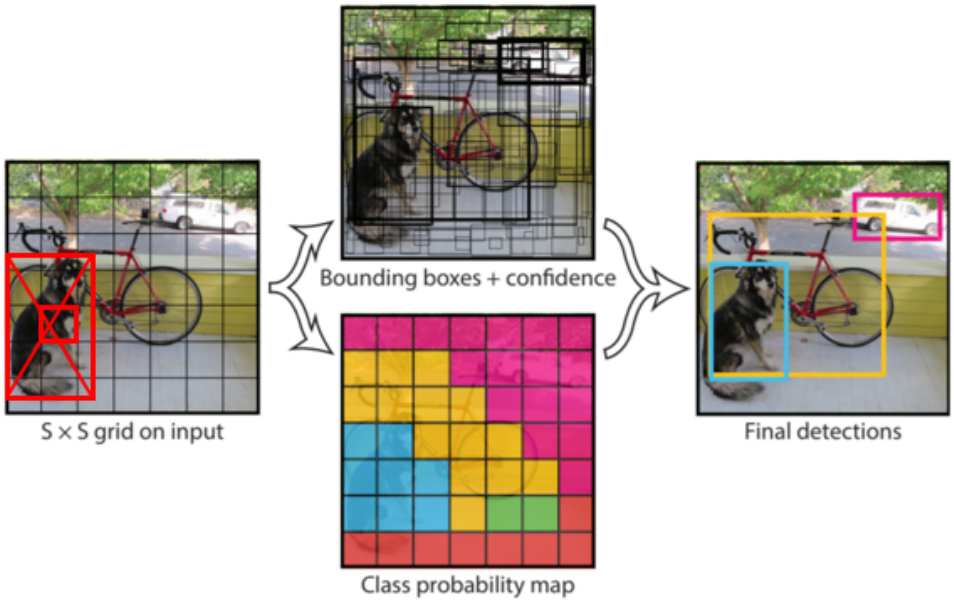


图1.3.2.1.1.1.1.1 YOLOv1 预测示意图

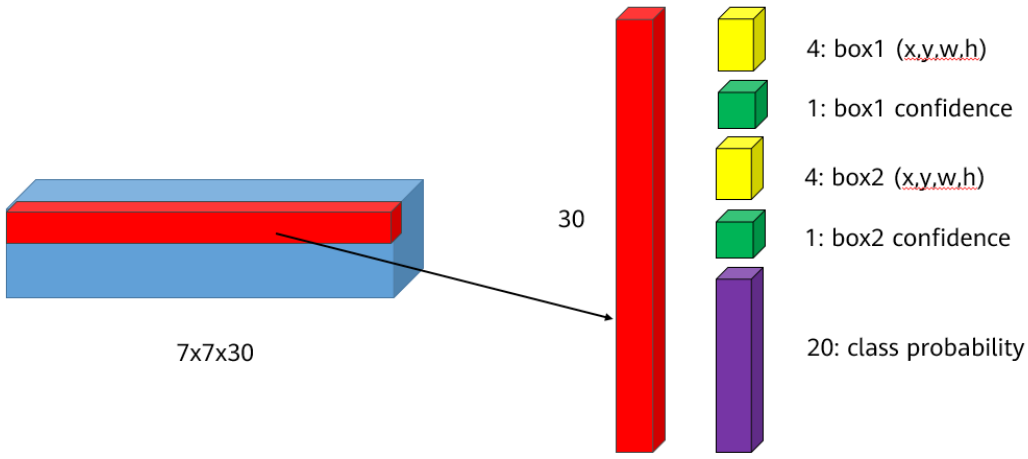


图1.3.2.1.1.1.2 YOLOv1 输出层

YOLOV2，选择了 5 个锚作为召回率和模型复杂度之间的良好折衷。其关键特点：

1）Batch Normalization: YOLOv1 没有使用 BN 层，而 YOLOv2 在每一层卷积层后都使用了 BN 层，BN 层通过训练数据学习每一层每个神经元的缩放比例，进行标准化。BN 层可以帮助网络进行训练，原论文指出，卷积层加了 BN 层后就可以不用 dropout 了，使用 BN 层后可以提高 2%的 mAP(平均准确率，mean average precision)。顺便一提的是，卷积层后加了 BN 层时，卷积层可以不使用偏置值。

2）High Resolution Classifier: 对 YOLOV2，预训练之后，在 ImageNet 数据集上，用 448*448 大小的图片对分类网络进行微调，大约 10 个 epoches，其目的是让网络先学习一下高分辨率的图片，之后再应用到检测网络中，这个举措使得 mAP 提升大概 4%。

3）Convolutional With Anchor Boxes: YOLOv1 并没有使用锚点，而是直接预测 x,y,w,h，且每个网格预测两个边界框的形式总觉得很奇怪（因为同一个网格中的两个边界框并没有什么不同）。而 YOLOv2 引用了 Faster RCNN 和 SSD 模型中的锚点，预测的位置是相对预置的锚点的。论文指出通过使用锚点，mAP 下降了 0.3%的 mAP，但是召回率增加了 7%，虽然 mAP 下降了，但是更高的召回率意味着模型的上限更高。

4）Dimension Cluster: 对网络来说，如果能够选择合适的 anchor 尺寸，网络更加容易学习并且预测出更好的结果，在论文中作者使用 k-means 算法在训练集上的边界框中自动选择合适的 box dimensions。

5）Direct location prediction: 作者在论文中提到，需要对 x,y,w,h 进行归一化（在输出层代表位置信息的部分使用 sigmoid 激活函数）。此外，置信度同样也需要进行归一化（输出层代码置信度的位置加 sigmoid 激活函数）。这样可以使得网络在训练过程中更加稳定。通过 Dimension Clusters 和 Direct location prediction 可以使模型提高 5%的 mAP。

6）Fine-Grained Features:在 13*13 特征图上进行目标检测，对于一些大的目标是足够的，但是对于小物体的检测还需要细粒度的特征，为此 YOLOV2 添加一个 passthrough layer，将浅层的特征和深层的特征，两个不同尺寸的特征按通道维度拼接起来。值得一提的是，原论文中，作者写的是 PassThrough 层将 26x26x512 的特征图变为 13x13x2048 的特征图，但是实际上，在作者的代码实现中，在 PassThrough 层前使用了 1x1 的卷积将 512 维的通道数减少到了 64 维。因此，实际上，PassThrough 层的输入为 26x26x64，输出为 13x13x256。而 YOLOv2 的最终网络结构如 8-3 所示。

7） Multi-Scale training：从上面的结构图可以看到，YOLOv2 相比 YOLOv1，去掉了全连接层，所有带参数的网络层均为卷积层和 BN 层（在表格中没画出来，每个卷积层后面都会跟一个 BN 层）。卷积层和 BN 层都不会受到输入图像大小的影响（如果网络有全连接层，输入

图像的大小必须是一致的）。因此，作者提出，在训练模型时可以使用不同尺度的图像进行训练来保证模型对大目标和小目标都能达到不错的效果。由于网络的输入图像的大小为输出大小的 32 倍，因此，作者使用了多个尺寸为 32 倍数的图像对网络进行训练。输入图像的大小为{320，352，...，608}，每十个 batch 换一组尺寸。

Type	Filters	Size	Output
Convolutional	32	3x3	416x416x32
MaxPooing		2x2/2	208x208x32
Convolutional	64	3x3	208x208x64
MaxPooing		2x2/2	104x104x64
Convolutional	128	3x3	104x104x128
Convolutional	64	1x1	104x104x64
Convolutional	128	3x3	104x104x128
MaxPooing		2x2/2	52x52x128
Convolutional	256	3x3	52x52x256
Convolutional	128	1x1	52x52x128
Convolutional	256	3x3	52x52x256
MaxPooing		2x2/2	26x26x256
Convolutional	512	3x3	26x26x512
Convolutional	256	1x1	26x26x256
Convolutional	512	3x3	26x26x512
Convolutional	256	1x1	26x26x256
Convolutional	512	3x3	26x26x512
MaxPooing		2x2/2	13x13x512
Convolutional	1024	3x3	13x13x1024
Convolutional	512	1x1	13x13x512
Convolutional	1024	3x3	13x13x1024
Convolutional	512	1x1	13x13x512
Convolutional	1024	3x3	13x13x1024

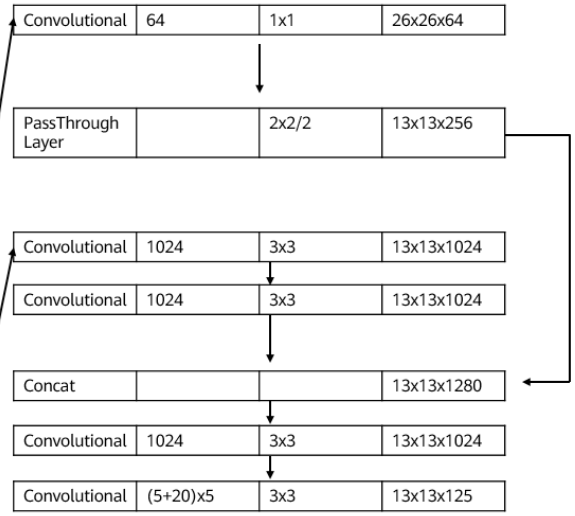


图1.3.2.1.1.1.3 YOLOv2 网络结构图

YOLOv3 相比 YOLOv2 最大的改进点在于借鉴了 SSD 的多尺度判别，即在不同大小的特征图上进行预测。对于网络前几层的大尺寸特征图，可以有效地检测出小目标，对于网络最后的小尺寸特征图可以有效地检测出大目标。此外，YOLOv3 的 backbone 选择了 DarkNet53 网络，网络结构更深，特征提取能力更强了。YOLOv3 的网络结构如下图所示，左侧中的红色框部分为去掉输出层的 DarkNet53 网络：

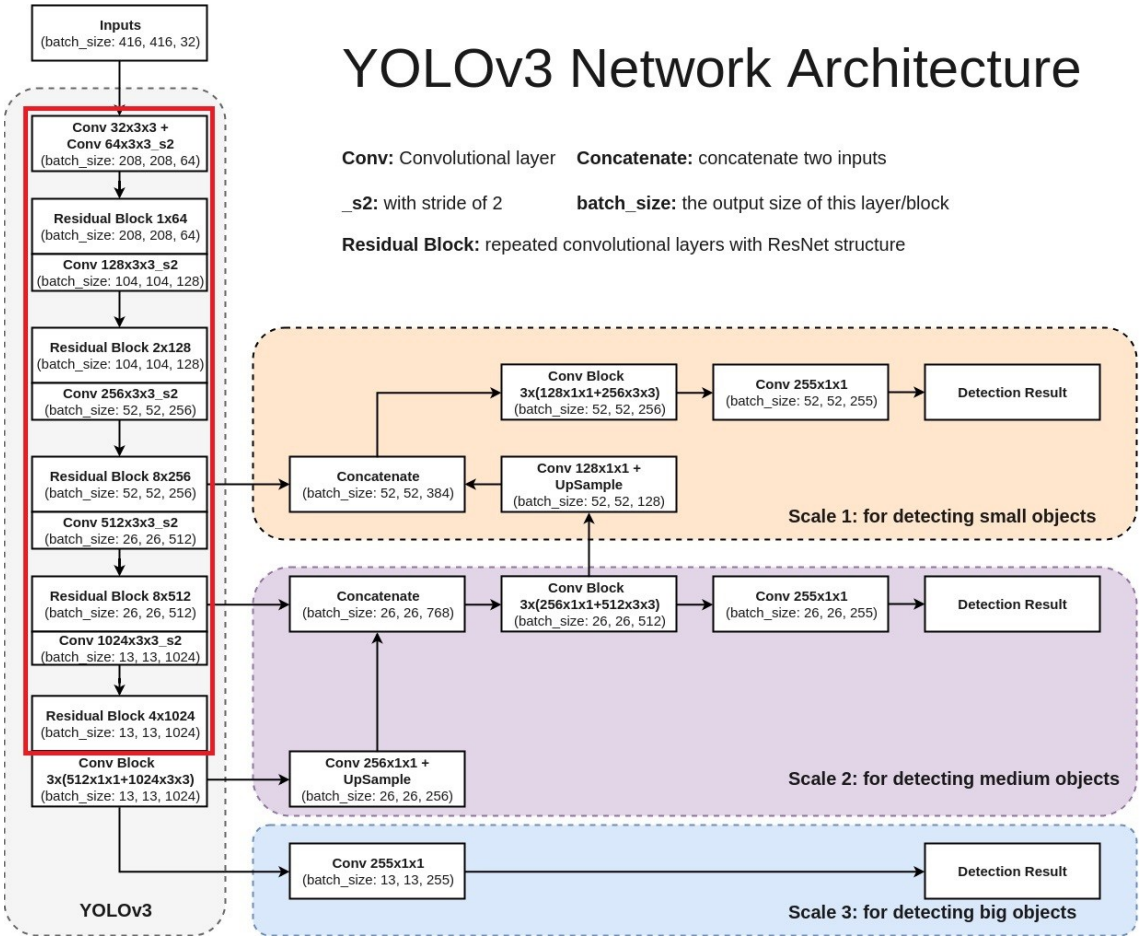


图1.3.2.1.1.1.4 YOLOv3 网络结构图

对于 YOLOv3 网络结构，有以下几点需要注意的：

1. 由于网络较深，使用了残差结构。
2. DarkNet53 网络用步长为 2 的卷积代替了池化层。
3. 所有的网络层不包含全连接层，因此，输入图像的大小也是可以调整的（有些地方看到的可能是 608x608，其实是一样的），而输入图像的大小同样是最小的输出特征图的 32 倍。
4. YOLOv3 分别在三个尺寸的特征图进行了预测，每个尺寸的特征图使用了 3 个锚点。因此，输出层的维度计算方法为：(4+1+80)x3=255，因此，最后一层 1x1 的卷积层的数量为 255。
5. 13 x 13 的特征图会通过上采样层和之前的 26 x 26 的特征图在通道维度拼接在一起，26 x 26 的特征图再经过上采样和 52 x 52 的特征图拼接。

1.4 MindSpore 训练

1.4.1 环境准备

本实验在华为云 ModelArts 的 Notebook 环境内运行。

请参考《MindSpore 环境搭建实验手册》的“4.1 ModelArts-Jupyter Notebook 开发环境搭建-训练用”部分。

1.4.2 项目介绍

项目文件结构：

```
├──code
   ├──main.ipynb        # 训练和测试入口文件
   ├──src
   │   ├──config.py          # 配置文件
   │   ├──dataset.py         # YOLOv3 网络定义文件配置文件
   │   ├──utils.py           # 工具类文件
   │   ├──yolov3.py          # 数据预处理文件
```

步骤 1　下载项目代码和数据集

在 Notebook 环境内新建 MindSpore 的内核环境，执行如下代码，将项目代码 code.zip 和数据集 data.zip 下载至云端环境内，并解压。

下载并解压项目代码：

```
!wget https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/deep-learning/code.zip
!unzip code.zip
```

下载并解压数据集：

```
!wget https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/deep-learning/data.zip
!unzip data.zip
```

*注意：Notebook 环境内上传、创建和编辑的文件均在/home/ma-user/work 目录下。

1.4.3 数据预处理

数据预处理文件在 code/src/dataset.py，无需执行。

数据预处理包括：

- 原始数据格式整理，将原始图片和 xml 标签处理为 mindrecord 格式；
- mindrecord 格式数据处理，将 mindrecord 格式的原始数据处理为网络需要的数据特征。

1.4.3.1 读取 xml 文件（dataset.py / xy_local）

```
"""YOLOv3 dataset"""
from __future__ import division

import os
from xml.dom.minidom import parse
import xml.dom.minidom

import numpy as np
from matplotlib.colors import rgb_to_hsv, hsv_to_rgb
from PIL import Image
import mindspore.dataset as de
from mindspore.mindrecord import FileWriter
import mindspore.dataset.vision.c_transforms as C
from src.config import ConfigYOLOV3ResNet18
```

```
# 读取 xml 文件标签，返回带有指定标签名的对象的集合
def xy_local(collection,element):
    xy = collection.getElementsByTagName(element)[0]
    xy = xy.childNodes[0].data
return xy
```

1.4.3.2 获取 annotation 标签（dataset.py / filter_valid_data）

```
def filter_valid_data(image_dir):
    """在 image_dir 和 anno_path 中过滤有效的图像"""

    label_id={'person':0, 'face':1, 'mask':2}
    all_files = os.listdir(image_dir)

    image_dict = {}
    image_files=[]
    for i in all_files:
        if (i[-3:]=='jpg' or i[-4:]=='jpeg') and i not in image_dict:
            image_files.append(i)
            label=[]
            xml_path = os.path.join(image_dir,i[:-3]+'xml')

            if not os.path.exists(xml_path):
                label=[[0,0,0,0,0]]
                image_dict[i]=label
                continue
            DOMTree = xml.dom.minidom.parse(xml_path)
            collection = DOMTree.documentElement
            # 在集合中获取所有框
            object_ = collection.getElementsByTagName("object")
            for m in object_:
                temp=[]
                name = m.getElementsByTagName('name')[0]
                class_num = label_id[name.childNodes[0].data]
                bndbox = m.getElementsByTagName('bndbox')[0]
                xmin = xy_local(bndbox,'xmin')
                ymin = xy_local(bndbox,'ymin')
                xmax = xy_local(bndbox,'xmax')
                ymax = xy_local(bndbox,'ymax')
                temp.append(int(xmin))
                temp.append(int(ymin))
                temp.append(int(xmax))
                temp.append(int(ymax))
                temp.append(class_num)
                label.append(temp)
            image_dict[i]=label
return image_files, image_dict
```

1.4.3.3 原始数据格式处理（dataset.py / data_to_mindrecord_byte_image）

dataset.py 中 data_to_mindrecord_byte_image 函数将原始数据处理为 mindrecord 格式。

字段解析如下所示：

- image: 原始图片；
- annotation: 标签，numpy 格式，维度：N*5，其中 N 代表框数量。5 分别表示：[xmin,ymin,xmax,ymax,class]；
- file: 图像文件名，为后期可视化存储原始图。

```
def data_to_mindrecord_byte_image(image_dir, mindrecord_dir, prefix, file_num):
    """通过 image_dir 和 anno_path 创建 MindRecord 文件"""
    mindrecord_path = os.path.join(mindrecord_dir, prefix)
    writer = FileWriter(mindrecord_path, file_num)
    image_files, image_anno_dict = filter_valid_data(image_dir)

    yolo_json = {
        "image": {"type": "bytes"},
        "annotation": {"type": "int32", "shape": [-1, 5]},
        "file": {"type": "string"},
    }
    writer.add_schema(yolo_json, "yolo_json")

    for image_name in image_files:
        image_path = os.path.join(image_dir, image_name)
        with open(image_path, 'rb') as f:
            img = f.read()
        annos = np.array(image_anno_dict[image_name],dtype=np.int32)
        #print(annos.shape)
        row = {"image": img, "annotation": annos, "file": image_name}
        writer.write_raw_data([row])
    writer.commit()
```

1.4.3.4 数据集预处理（dataset.py / preprocess_fn）

mindrecord 数据预处理过程如下所示：

- 图片和框裁剪、resize 到网络设定输入图片尺寸([352,640])得到 images
- 求框和锚点之间的 iou 值，将框和锚点对应。
- 将框、可信度、类别对应在网格中，得到 bbox_1、bbox_2、bbox_3。
- 将网格中框单独拿出来，得到 gt_box1、gt_box2、gt_box3。

*注意：

- 本实验一张图片最多框数量设定为 50，故最多可以保存 50 个框。

- 本实验网格采用大（ 32×32 ）、中（ 16×16 ）、小（ 8×8 ）网格。选取 9 组锚点，其中包括 3 个大框、3 个中框、3 个小框。3 个大框锚点采用大网格映射，3 个中框采用中网格映射，3 个小框采用小网格映射。
- 训练图片采用了数据增强，增强方式包括：随机裁剪噪声、翻转、扭曲。

本实验将所有图片都统一为[352,640]进行训练和推理。

图片和框处理参照 dataset.py / preprocess_fn / def _infer_data。

测试图片改变尺寸使用传统的先裁剪后 resize 方式，在保证图片不失真的情况下改变图片尺寸。

代码解析：

- 变量 scale 为真实边和设定边（[352,640]）的最小比例。为了保证图片不失真，长宽采取相同的放缩比例。
- 将图片 resize 为(nw, nh)后，预设定图片尺寸（[352,640]）不同，需要进行填充。填充采用两边填充。即将 resize 后的(nw, nh)大小图片放在（[352,640]）中间，两边填充像素值 128。
- 测试框不需要做任何处理。

训练图片处理 dataset.py / preprocess_fn / def _data_aug。

训练图片改变尺寸在传统的先裁剪后 resize 方式的基础上添加了随机噪声。使图片在失真范围内带有更多的原始图片信息。

代码解析：

- 变量 h,w 为设定图片大小（[352,640]），变量 iw, ih 为原始图片大小。
- 变量 scale 为尺度的意思，在（0.25，2）比例尺度范围内获取多尺度特征。得到新的图片大小 nh 或者 nw，nh 和 nw 想不与 h，w 带有不同尺度特征。
- 变量 jitter 控制随机噪声大小。变量 new_ar 定义如下所示，在宽高比 $\frac{float(w)}{float(h)}$ 的基础上乘以一个 1 左右的随机数。随机数为 $\frac{¿rand(1-jitter,1+jitter)}{¿rand(1-jitter,1+jitter)}$ 。通过改变 new_ar 从而改变 resize 后的图片大小(nw, nh)。带有噪声的图片(nw, nh)在一定范围内失真，失真比例由 jitter 控制

$$new_{ar}=\frac{float(w)}{float(h)}\times\frac{¿rand(1-jitter,1+jitter)}{¿rand(1-jitter,1+jitter)}$$

- 变量 dx 和 dy 代表噪声大小。
- 对图片和框进行相同的 resize 操作。
- 操作方式为将大小为(nw, nh)的图片填充到（w,h）的 dx,dy 位置。其他位置用 128 填充。
- box_data 维度为[50,5]，其中 50 代表每张图片最多框数设定。5 代表[xmin,ymin,xmax,ymax,class] 。通过修改 config.py 文件的 nms_max_num 大小，可以修改最大框数量。但是最大框数量必须比所有真实数据图片的最大框数大，否则制作数据集过程会引起 box 数量过多报错。

- 得到的变量 images 为预处理结果，可以直接输入网络训练。得到的变量 box_data 需要进一步传入函数_preprocess_true_boxes 中进行锚点和网格映射，具体参考下一步框预处理。

训练框预处理（框、锚点、网格映射，bbox 计算）dataset.py / preprocess_fn / def _preprocess_true_boxes。

代码解析：

- true_boxes 为上一步结果 box_data，具体请参照上一个解析。
- num_layers 为锚点层数，本实验，将锚点分为三个层次。分别为大框，中框，小框。
- anchor_mask 为锚点编号。config.py 中的 anchor_scales 为锚点。根据标签框的大小来设定。
- boxes_xy 代表框的中心点坐标，boxes_wh 代表框的宽高。
- grid_shapes 为网格，其中包含大网格（大框使用），中网格（中框使用），小网格（小框使用）。大框尺寸为 32×32，中框尺寸为 16×16，小框尺寸为 8×8。网格维度分别为大框[11,20]，中框[22,40]，小框[44,80]。（注：$11=\frac{352}{32}$，$20=\frac{640}{32}$，$22=\frac{640}{20}$，$40=\frac{640}{16}$，$44=\frac{352}{8}$，$11=\frac{640}{8}$）
- y_true 为已经映射到锚点、网格的框。y_true[0]为大框映射，维度为[11, 20, 3, 8]，y_true[1]为中框映射，维度为[22, 40, 3, 8]， y_true[2]为小框映射，维度为[44, 80, 3, 8]。 第 2 个维度 3 代表每个层次有几个框。本实验大框、中框、小框层都是有三个框对应。第 3 个维度 8 代表标签值，分别代表[boxes_xy,boxes_wh, Confidence(置信度)，class(one-hot 值，person、face、mask)]
- 映射方式如下所示。

框映射到锚点和网格流程如下：

1. 将输入 boxes_xy 和 boxes_wh 的值放缩到 0-1 范围内。
2. 将锚点中心点放缩到（0，0）点，并扩展维度。最后得到 anchors_max 和 anchors_min 维度为[1,9,2]，其中 anchors_min 代表框中心点放缩到（0，0）点以后的左下角坐标。anchors_max 代表框中心点放缩到（0，0）点以后的右上角坐标。
3. 将 boxes_wh 中心点放缩到（0，0）点，并扩展维度。最后得到 boxes_max 和 boxes_min 维度为[num_box,1,2]。其中 boxes_min 代表框中心点放缩到（0，0）点以后的左下角坐标。boxes_max 代表框中心点放缩到（0，0）点以后的右上角坐标。
4. 分别求每个框和 9 个锚点的交集宽和高。具体求法为：
   - 计算 anchors_min 和 anchors_min 的最大值 intersect_min，intersect_min 为交集左下角坐标
   - 计算 anchors_max 和 boxes_max 的最小值 intersect_max，intersect_max 为交集右上角坐标
   - 计算交集宽高为：intersect_wh = intersect_max - intersect_min
5. 分别求每个框和 9 个锚点的 iou 值。iou 为交并比。其中交集面积为 intersect_area，并集面积为(box_area + anchor_area - intersect_area)。iou 维度为[num_box, 9]，其中 9 代表 9 个锚点。
6. 对于每个框，选出最大 iou 值对应的锚点编号。

7. 将 true_boxes 映射到锚点编号、网格中。其中 y_true 的解析参考前面解析。y_true 即数据预处理结果 bbox_1、bbox_2、bbox_3。矩阵数据范围为 0-1（参考步骤 1 放缩）。
    - 将锚点和真实框放缩到（0，1）的目标是保证其中心点统一。锚点是没有中心点的。所有网格点都有锚点。这种放缩可以快速定位目标框属于哪个锚点（网格化以后的锚点，以小尺度为例子有 11×20×3 个锚点）。

框预处理（框、锚点、网格映射，gt_box 计算）dataset.py / preprocess_fn / def _preprocess_true_boxes。

代码解析：

- gt_box 维度为[50,4]，存放了所有的 y_true 中的框，忽略网格和锚点信息。gt_box 即数据预处理结果 gt_box1、gt_box2、gt_box3。矩阵数据范围为 0-1。

详细代码如下：

```
def preprocess_fn(image, box, file, is_training):
    """数据集预处理函数."""
    config_anchors = []
    temp = ConfigYOLOV3ResNet18.anchor_scales
    for i in temp:
        config_anchors+=list(i)

    anchors = np.array([float(x) for x in config_anchors]).reshape(-1, 2)
    do_hsv = False
    max_boxes = ConfigYOLOV3ResNet18._NUM_BOXES   # 最多框，50
    num_classes = ConfigYOLOV3ResNet18.num_classes   # 类别数，3

    # 随机数
    def _rand(a=0., b=1.):
        return np.random.rand() * (b - a) + a

    # true_boxes 为函数_data_aug 的结果 box_data
    def _preprocess_true_boxes(true_boxes, anchors, in_shape=None):
        """获取 True 边界框"""
        # num_layers 为锚点层数，本实验将锚点分为大框、中框、小框三个层次
        num_layers = anchors.shape[0] // 3
        # anchor_mask 为锚点编号
        anchor_mask = [[6, 7, 8], [3, 4, 5], [0, 1, 2]]
        true_boxes = np.array(true_boxes, dtype='float32')
        input_shape = np.array(in_shape, dtype='int32')

        # boxes_xy 代表框的中心点坐标，boxes_wh 代表框的宽高
        boxes_xy = (true_boxes[..., 0:2] + true_boxes[..., 2:4]) // 2.
        boxes_wh = true_boxes[..., 2:4] - true_boxes[..., 0:2]

        # 将输入 boxes_xy 和 boxes_wh 的值放缩到 0-1 范围内
        true_boxes[..., 0:2] = boxes_xy / input_shape[::-1]
        true_boxes[..., 2:4] = boxes_wh / input_shape[::-1]

        # grid_shapes 为网格，包含大网格（大框使用），中网格（中框使用），小网格（小框使用）
```

```
# 大框尺寸为 32×32，中框尺寸为 16×16，小框尺寸为 8×8
# 网格维度分别为大框[11,20]，中框[22,40]，小框[44,80]
      grid_shapes = [input_shape // 32, input_shape // 16, input_shape // 8]

# y_true 为已经映射到锚点、网格的框
# y_true[0]为大框映射，维度为[11, 20, 3, 8]，y_true[1]为中框映射，维度为[22, 40, 3, 8]，
y_true[2]为小框映射，维度为[44, 80, 3, 8]
# 其中 3 代表每个层次有几个框，本实验大框、中框、小框层都是有三个框对应
# 其中 8 代表标签值，分别代表[boxes_xy,boxes_wh, Confidence(置信度)，class(one-hot 值，
person、face、mask)]
      y_true = [np.zeros((grid_shapes[l][0], grid_shapes[l][1], len(anchor_mask[l]),
                          5 + num_classes), dtype='float32') for l in range(num_layers)]



      # 将锚点中心点放缩到（0，0）点，并扩展维度
      anchors = np.expand_dims(anchors, 0)
# 得到 anchors_max 和 anchors_min，维度为[1,9,2]
# anchors_max 代表框中心点放缩到（0，0）点以后的右上角坐标
      anchors_max = anchors / 2.
# anchors_min 代表框中心点放缩到（0，0）点以后的左下角坐标
      anchors_min = -anchors_max
      # 将 boxes_wh 中心点放缩到（0，0 点）
      valid_mask = boxes_wh[..., 0] >= 1
      wh = boxes_wh[valid_mask]
      if len(wh) >= 1:
         # 扩展维度
         wh = np.expand_dims(wh, -2)
         # 得到 boxes_max 和 boxes_min 维度为[num_box,1,2]
# boxes_max 代表框中心点放缩到（0，0）点以后的右上角坐标
         boxes_max = wh / 2.
# boxes_min 代表框中心点放缩到（0，0）点以后的左下角坐标
         boxes_min = -boxes_max


# 分别求每个框和 9 个锚点的 iou 值
# iou 为交并比，iou 维度为[num_box, 9]，其中 9 代表 9 个锚点
         intersect_min = np.maximum(boxes_min, anchors_min)
         intersect_max = np.minimum(boxes_max, anchors_max)
         intersect_wh = np.maximum(intersect_max - intersect_min, 0.)
         # intersect_area 为交集面积
         intersect_area = intersect_wh[..., 0] * intersect_wh[..., 1]
         box_area = wh[..., 0] * wh[..., 1]
         anchor_area = anchors[..., 0] * anchors[..., 1]
         # 并集面积为(box_area + anchor_area - intersect_area)
         iou = intersect_area / (box_area + anchor_area - intersect_area)
         # 对于每个框，选出最大 iou 值对应的锚点编号
         best_anchor = np.argmax(iou, axis=-1)
         # 将 true_boxes 映射到锚点编号、网格中
```

```
            for t, n in enumerate(best_anchor):
                for l in range(num_layers):
                    if n in anchor_mask[l]:
                        # 将锚点和真实框放缩到（0，1），目的是保证其中心点统一
                        i = np.floor(true_boxes[t, 0] * grid_shapes[l][1]).astype('int32')
                        j = np.floor(true_boxes[t, 1] * grid_shapes[l][0]).astype('int32')
                        # 可以快速定位目标框属于哪个锚点
                        k = anchor_mask[l].index(n)

                        c = true_boxes[t, 4].astype('int32')
                        # y_true 即数据预处理结果 bbox_1、bbox_2、bbox_3，矩阵数据范围为 0-1
                        y_true[l][j, i, k, 0:4] = true_boxes[t, 0:4]
                        y_true[l][j, i, k, 4] = 1.
                        y_true[l][j, i, k, 5 + c] = 1.

        pad_gt_box0 = np.zeros(shape=[ConfigYOLOV3ResNet18._NUM_BOXES, 4],
dtype=np.float32)
        pad_gt_box1 = np.zeros(shape=[ConfigYOLOV3ResNet18._NUM_BOXES, 4],
dtype=np.float32)
        pad_gt_box2 = np.zeros(shape=[ConfigYOLOV3ResNet18._NUM_BOXES, 4],
dtype=np.float32)

# gt_box 维度为[50,4]，存放了所有的 y_true 中的框，忽略网格和锚点信息
# gt_box 即数据预处理结果 gt_box1、gt_box2、gt_box3，矩阵数据范围为 0-1。
        mask0 = np.reshape(y_true[0][..., 4:5], [-1])
        gt_box0 = np.reshape(y_true[0][..., 0:4], [-1, 4])
        gt_box0 = gt_box0[mask0 == 1]
        pad_gt_box0[:gt_box0.shape[0]] = gt_box0

        mask1 = np.reshape(y_true[1][..., 4:5], [-1])
        gt_box1 = np.reshape(y_true[1][..., 0:4], [-1, 4])
        gt_box1 = gt_box1[mask1 == 1]
        pad_gt_box1[:gt_box1.shape[0]] = gt_box1

        mask2 = np.reshape(y_true[2][..., 4:5], [-1])
        gt_box2 = np.reshape(y_true[2][..., 0:4], [-1, 4])
        gt_box2 = gt_box2[mask2 == 1]
        pad_gt_box2[:gt_box2.shape[0]] = gt_box2

        return y_true[0], y_true[1], y_true[2], pad_gt_box0, pad_gt_box1, pad_gt_box2

# 图片和框 resize，本实验将图片都统一为[352,640]进行训练和推理
# 测试图片改变尺寸使用传统的先裁剪后 resize 方式，在保证图片不失真的情况下改变图片尺寸
    def _infer_data(img_data, input_shape, box):
        w, h = img_data.size # 真实边
        input_h, input_w = input_shape  # 设定边（[352,640]）
            # 变量 scale 为真实边和设定边的最小比例。为了保证图片不失真，长宽采取相同的放缩比例。
        scale = min(float(input_w) / float(w), float(input_h) / float(h))
```

```
        nw = int(w * scale)
        nh = int(h * scale)
        img_data = img_data.resize((nw, nh), Image.BICUBIC)

            # 将图片 resize 为(nw, nh)后，与预设定图片尺寸（[352,640]）不同，需要进行填充。
        new_image = np.zeros((input_h, input_w, 3), np.float32)

            # 填充采用两边填充，即将 resize 后的(nw, nh)大小图片放在（[352,640]）中间
# 两边填充像素值 128
        new_image.fill(128)
        img_data = np.array(img_data)
        if len(img_data.shape) == 2:
            img_data = np.expand_dims(img_data, axis=-1)
            img_data = np.concatenate([img_data, img_data, img_data], axis=-1)

        dh = int((input_h - nh) / 2)
        dw = int((input_w - nw) / 2)
        new_image[dh:(nh + dh), dw:(nw + dw), :] = img_data
        new_image /= 255.
        new_image = np.transpose(new_image, (2, 0, 1))
        new_image = np.expand_dims(new_image, 0)
            # 测试框不需要做任何处理
        return new_image, np.array([h, w], np.float32), box

    def _data_aug(image, box, is_training, jitter=0.3, hue=0.1, sat=1.5, val=1.5,
image_size=(352, 640)):

        """数据增强函数."""
        if not isinstance(image, Image.Image):
            image = Image.fromarray(image)

        # 变量 iw, ih 为原始图片大小
        iw, ih = image.size
        ori_image_shape = np.array([ih, iw], np.int32)
        # 变量 h,w 为设定图片大小（[352,640]）
        h, w = image_size

        if not is_training:
            return _infer_data(image, image_size, box)

        flip = _rand() < .5

# box_data 维度为[50,5]
# 其中 50 代表每张图片最多框数设定，5 代表[xmin,ymin,xmax,ymax,class]
        box_data = np.zeros((max_boxes, 5))
        flag =0

        while True:
```

```
        # 防止所有框都被清除
        # 变量 jitter 控制随机噪声大小
# 带有噪声的图片(nw, nh)在一定范围内失真，失真比例由 jitter 控制
        new_ar = float(w) / float(h) * _rand(1 - jitter, 1 + jitter) / \
                _rand(1 - jitter, 1 + jitter)
        # 变量 scale 为尺度的意思，在（0.25，2）比例尺度范围内获取多尺度特征
        scale = _rand(0.25, 2)

        # 通过改变 new_ar 从而改变 resize 后的图片大小(nw, nh)
        if new_ar < 1:
            nh = int(scale * h)
            nw = int(nh * new_ar)
        else:
            nw = int(scale * w)
            nh = int(nw / new_ar)

        # 变量 dx 和 dy 代表噪声大小
        dx = int(_rand(0, w - nw))
        dy = int(_rand(0, h - nh))
        flag = flag + 1

        # 对图片和框进行相同的 resize 操作
         # 将大小为(nw, nh)的图片填充到（w,h）的 dx,dy 位置
        if len(box) >= 1:
            t_box = box.copy()
            np.random.shuffle(t_box)
            t_box[:, [0, 2]] = t_box[:, [0, 2]] * float(nw) / float(iw) + dx
            t_box[:, [1, 3]] = t_box[:, [1, 3]] * float(nh) / float(ih) + dy
            if flip:
                t_box[:, [0, 2]] = w - t_box[:, [2, 0]]
            t_box[:, 0:2][t_box[:, 0:2] < 0] = 0
            t_box[:, 2][t_box[:, 2] > w] = w
            t_box[:, 3][t_box[:, 3] > h] = h
            box_w = t_box[:, 2] - t_box[:, 0]
            box_h = t_box[:, 3] - t_box[:, 1]
            t_box = t_box[np.logical_and(box_w > 1, box_h > 1)]  # 去掉无效框

        if len(t_box) >= 1:
            box = t_box
            break

    # 得到的变量 box_data 需要进一步传入函数_preprocess_true_boxes 中进行锚点和网格映射
    box_data[:len(box)] = box
    # 调整图像大小
    image = image.resize((nw, nh), Image.BICUBIC)
    # 替换图像，其他位置用 128 填充
    new_image = Image.new('RGB', (w, h), (128, 128, 128))
```

```
        new_image.paste(image, (dx, dy))
        # 得到的变量 images 为预处理结果，可以直接输入网络训练
        image = new_image

        # 是否翻转图像
        if flip:
            image = image.transpose(Image.FLIP_LEFT_RIGHT)

        # 是否转灰度图
        gray = _rand() < .25
        if gray:
            image = image.convert('L').convert('RGB')

        # 当图像通道数为 1
        image = np.array(image)
        if len(image.shape) == 2:
            image = np.expand_dims(image, axis=-1)
            image = np.concatenate([image, image, image], axis=-1)

        # 扭曲图像
        hue = _rand(-hue, hue)
        sat = _rand(1, sat) if _rand() < .5 else 1 / _rand(1, sat)
        val = _rand(1, val) if _rand() < .5 else 1 / _rand(1, val)
        image_data = image / 255.
        if do_hsv:
            x = rgb_to_hsv(image_data)
            x[..., 0] += hue
            x[..., 0][x[..., 0] > 1] -= 1
            x[..., 0][x[..., 0] < 0] += 1
            x[..., 1] *= sat
            x[..., 2] *= val
            x[x > 1] = 1
            x[x < 0] = 0
            image_data = hsv_to_rgb(x)  # numpy array, 0 to 1
        image_data = image_data.astype(np.float32)

        # 预处理边界框
        bbox_true_1, bbox_true_2, bbox_true_3, gt_box1, gt_box2, gt_box3 = \
            _preprocess_true_boxes(box_data, anchors, image_size)

        return image_data, bbox_true_1, bbox_true_2, bbox_true_3, \
               ori_image_shape, gt_box1, gt_box2, gt_box3

    if is_training:
        images, bbox_1, bbox_2, bbox_3, image_shape, gt_box1, gt_box2, gt_box3 =
_data_aug(image, box, is_training)
        return images, bbox_1, bbox_2, bbox_3, gt_box1, gt_box2, gt_box3
```

```
    images, shape, anno = _data_aug(image, box, is_training)
    return images, shape, anno, file
```

1.4.3.5 使用 MindDataset 创建 YOLOv3 数据（dataset.py / create_yolo_dataset）

```
def create_yolo_dataset(mindrecord_dir, batch_size=32, repeat_num=1, device_num=1, rank=0,
                        is_training=True, num_parallel_workers=8):
    """使用 MindDataset 创建 YOLOv3 数据集"""
    ds = de.MindDataset(mindrecord_dir, columns_list=["image", "annotation","file"], num_shards=device_num, shard_id=rank,
                        num_parallel_workers=num_parallel_workers, shuffle=is_training)
    decode = C.Decode()
    ds = ds.map(operations=decode, input_columns=["image"])
    compose_map_func = (lambda image, annotation, file: preprocess_fn(image, annotation,file, is_training))

    if is_training:
        hwc_to_chw = C.HWC2CHW()
        ds = ds.map(operations=compose_map_func, input_columns=["image", "annotation","file"],
                    output_columns=["image", "bbox_1", "bbox_2", "bbox_3", "gt_box1", "gt_box2", "gt_box3"],
                    column_order=["image", "bbox_1", "bbox_2", "bbox_3", "gt_box1", "gt_box2", "gt_box3"],
                    num_parallel_workers=num_parallel_workers)
        ds = ds.map(operations=hwc_to_chw, input_columns=["image"], num_parallel_workers=num_parallel_workers)
        ds = ds.batch(batch_size, drop_remainder=True)
        ds = ds.repeat(repeat_num)
    else:
        ds = ds.map(operations=compose_map_func, input_columns=["image", "annotation","file"],
                    output_columns=["image", "image_shape", "annotation","file"],
                    column_order=["image", "image_shape", "annotation","file"],
                    num_parallel_workers=num_parallel_workers)
    return ds
```

1.4.3.6 预处理结果展示

数据预处理结果可以直接用于训练和推理。预处理结果解析如下所示。

网络训练输入数据（训练预处理结果）解析：

名称	维度	描述
images	(32, 3, 352, 640)	图片[batch_size,channel,weight,height]
bbox_1	(11, 20, 3, 8)	大框在大尺度（32*32）映射 [grid_big,grid_big,num_big,label]

bbox_2	(22, 40, 3, 8)	中框在中尺度（16*16）映射 [grid_middle,grid_big,num_middle,label]
bbox_3	(44, 80, 3, 8)	小框在小尺度（8*8）映射 [grid_small,grid_small,num_small,label]
gt_box1	(50, 4)	大框
gt_box2	(50, 4)	中框
gt_box3	(50, 4)	小框

表1.4.3.6.1.1.1.1.1

网络测试输入数据（测试预处理结果）解析：

名称	维度及说明
images	图片：(1, 3, 352, 640)
shape	图片尺寸，例如：（ 720P、 1280P）
anno	真实框[xmin,ymin,xmax,ymax]

表1.4.3.6.1.1.1.1.2

1.4.4 YOLOv3 网络定义

YOLOv3 网络定义文件在 code/src/yolov3.py，无需执行，但“定义 YOLOv3 模块“处设计了挖空需补全代码。

为了让模型简单，我们选用 ResNet-18 作为我们的主干网络。

- 定义 ResNet18 主干网络
- 定义 YOLOv3 网络
- 定义 IoU
- 定义 loss 计算
- YOLOv3 验证网络结构

1.4.4.1 定义 ResNet18 网络

本实验使用 ResNet18 网络提取特征，分别提取到特征 feature_map1、feature_map2、feature_map3。

输入输出变量分析：

名称	维度	描述
输入：x	(32, 3, 352, 640)	网络输入图片 [batch_size,channel,weight,height]
resnet 输出：feature_map1	(32, 128, 44, 80)	大尺度特征[batch_size, backbone_shape[2], h/8, w/8]
resnet 输出：feature_map2	(32, 256, 22, 40)	中尺度特征 [batch_size, backbone_shape[3], h/16, w/16]

resnet 输出：feature_map3	(32, 512, 11, 20)	小尺度特征[batch_size, backbone_shape[4], h/32, w/32]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

图1.4.4.1.1.1.1.1 ResNet18 网络结构

```
"""基于 ResNet18 的 YOLOv3"""

import numpy as np
import mindspore as ms
import mindspore.nn as nn
from mindspore import context, Tensor
from mindspore.context import ParallelMode
from mindspore.parallel._auto_parallel_context import auto_parallel_context
from mindspore.communication.management import get_group_size
from mindspore.common.initializer import TruncatedNormal
from mindspore.ops import operations as P
from mindspore.ops import functional as F
from mindspore.ops import composite as C


def weight_variable():
    """权重变量"""
    return TruncatedNormal(0.02)

# 卷积
class _conv2d(nn.Cell):
    """Create Conv2D with padding."""
    def __init__(self, in_channels, out_channels, kernel_size, stride=1):
        super(_conv2d, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels,
                              kernel_size=kernel_size, stride=stride, padding=0, pad_mode='same',
                              weight_init=weight_variable())
```

```
    def construct(self, x):
        x = self.conv(x)
        return x


def _fused_bn(channels, momentum=0.99):
    """批次规范化"""
    return nn.BatchNorm2d(channels, momentum=momentum)


def _conv_bn_relu(in_channel,
                  out_channel,
                  ksize,
                  stride=1,
                  padding=0,
                  dilation=1,
                  alpha=0.1,
                  momentum=0.99,
                  pad_mode="same"):
    """获得 conv2d batchnorm 和 relu 层"""
    return nn.SequentialCell(
        [nn.Conv2d(in_channel,
                   out_channel,
                   kernel_size=ksize,
                   stride=stride,
                   padding=padding,
                   dilation=dilation,
                   pad_mode=pad_mode),
         nn.BatchNorm2d(out_channel, momentum=momentum),
         nn.LeakyReLU(alpha)]
    )

# 定义残差块
class BasicBlock(nn.Cell):
    """
    ResNet basic block.

    Args:
        in_channels (int): Input channel.
        out_channels (int): Output channel.
        stride (int): Stride size for the initial convolutional layer. Default:1.
        momentum (float): Momentum for batchnorm layer. Default:0.1.

    Returns:
        Tensor, output tensor.

    Examples:
```

```
        BasicBlock(3,256,stride=2,down_sample=True).
    """
    expansion = 1

    def __init__(self,
                 in_channels,
                 out_channels,
                 stride=1,
                 momentum=0.99):
        super(BasicBlock, self).__init__()

        self.conv1 = _conv2d(in_channels, out_channels, 3, stride=stride)
        self.bn1 = _fused_bn(out_channels, momentum=momentum)
        self.conv2 = _conv2d(out_channels, out_channels, 3)
        self.bn2 = _fused_bn(out_channels, momentum=momentum)
        self.relu = P.ReLU()
        self.down_sample_layer = None
        self.downsample = (in_channels != out_channels)
        if self.downsample:
            self.down_sample_layer = _conv2d(in_channels, out_channels, 1, stride=stride)
        self.add = P.TensorAdd()

    def construct(self, x):
        identity = x

        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)

        x = self.conv2(x)
        x = self.bn2(x)

        if self.downsample:
            identity = self.down_sample_layer(identity)

        out = self.add(x, identity)
        out = self.relu(out)

        return out

# 定义残差网络
class ResNet(nn.Cell):
    """
    ResNet network.

    Args:
        block (Cell): Block for network.
```

```
    layer_nums (list): Numbers of different layers.
    in_channels (int): Input channel.
    out_channels (int): Output channel.
    num_classes (int): Class number. Default:100.

Returns:
    Tensor, output tensor.

Examples:
    ResNet(ResidualBlock,
           [3, 4, 6, 3],
           [64, 256, 512, 1024],
           [256, 512, 1024, 2048],
           100).
"""

def __init__(self,
             block,
             layer_nums,
             in_channels,
             out_channels,
             strides=None,
             num_classes=80):
    super(ResNet, self).__init__()

    if not len(layer_nums) == len(in_channels) == len(out_channels) == 4:
        raise ValueError("the length of "
                         "layer_num, inchannel, outchannel list must be 4!")

    self.conv1 = _conv2d(3, 64, 7, stride=2)
    self.bn1 = _fused_bn(64)
    self.relu = P.ReLU()
    self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, pad_mode='same')

    self.layer1 = self._make_layer(block,
                                   layer_nums[0],
                                   in_channel=in_channels[0],
                                   out_channel=out_channels[0],
                                   stride=strides[0])
    self.layer2 = self._make_layer(block,
                                   layer_nums[1],
                                   in_channel=in_channels[1],
                                   out_channel=out_channels[1],
                                   stride=strides[1])
    self.layer3 = self._make_layer(block,
                                   layer_nums[2],
                                   in_channel=in_channels[2],
```

```
                                out_channel=out_channels[2],
                                stride=strides[2])
        self.layer4 = self._make_layer(block,
                                layer_nums[3],
                                in_channel=in_channels[3],
                                out_channel=out_channels[3],
                                stride=strides[3])

        self.num_classes = num_classes
        if num_classes:
            self.reduce_mean = P.ReduceMean(keep_dims=True)
            self.end_point = nn.Dense(out_channels[3], num_classes, has_bias=True,
                                      weight_init=weight_variable(),
                                      bias_init=weight_variable())
            self.squeeze = P.Squeeze(axis=(2, 3))
    # 创建 ResNet 的层
    def _make_layer(self, block, layer_num, in_channel, out_channel, stride):
        """
        Make Layer for ResNet.

        Args:
            block (Cell): Resnet block.
            layer_num (int): Layer number.
            in_channel (int): Input channel.
            out_channel (int): Output channel.
            stride (int): Stride size for the initial convolutional layer.

        Returns:
            SequentialCell, the output layer.

        Examples:
            _make_layer(BasicBlock, 3, 128, 256, 2).
        """
        layers = []

        resblk = block(in_channel, out_channel, stride=stride)
        layers.append(resblk)

        for _ in range(1, layer_num - 1):
            resblk = block(out_channel, out_channel, stride=1)
            layers.append(resblk)

        resblk = block(out_channel, out_channel, stride=1)
        layers.append(resblk)

        return nn.SequentialCell(layers)
```

```
    def construct(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        c1 = self.maxpool(x)

        c2 = self.layer1(c1)
        c3 = self.layer2(c2)
        c4 = self.layer3(c3)
        c5 = self.layer4(c4)

        out = c5
        if self.num_classes:
            out = self.reduce_mean(c5, (2, 3))
            out = self.squeeze(out)
            out = self.end_point(out)

        return c3, c4, out

# 定义 ResNet18 残差网络
def resnet18(class_num=10):
    """
    Get ResNet18 neural network.

    Args:
        class_num (int): Class number.

    Returns:
        Cell, cell instance of ResNet18 neural network.

    Examples:
        resnet18(100).
    """
    return ResNet(BasicBlock,
                  [2, 2, 2, 2],
                  [64, 64, 128, 256],
                  [64, 128, 256, 512],
                  [1, 2, 2, 2],
                  num_classes=class_num)
```

1.4.4.2 定义 YOLOv3 网络

确定了 backbone 部分之后，接下来，我们就可以搭建 YOLOv3 网络了。

本实验使用卷积网络和上采样将特征和锚点对应，得到 big_object_output、medium_object_output、small_object_output。

输入输出变量分析：

名称	维度	描述

输出：big_object_output	(32, 24, 11, 20)	输出小尺度特征[batch_size, out_channel, h/32, w/32]
输出：medium_object_output	(32, 24, 22, 40)	输出中尺度特征 [batch_size, out_channel, h/16, w/16]
输出：small_object_output	(32, 24, 44, 80)	输出大尺度特征[batch_size, out_channel, h/8, w/8]

表1.4.4.2.1.1.1.1.1

本实验 out_channel=24，计算方式为：

$$out_{channel}=\frac{len(anchor_{scales})}{3}\times(num_{classes}+5)$$

这里的 num_classes 代表 8 个标签，分别为 4 个位置信息：框的中心点坐标和框的长宽；一个置信度：框的概率；类别信息：共三类。

$\frac{len(anchor_{scales})}{3}$ 代表每种尺度锚点个数，本实验每种尺度包含 3 个锚点。

```
# 定义 YOLOv3 模块
class YoloBlock(nn.Cell):
    """
    YoloBlock for YOLOv3.
    Args:
        in_channels (int): Input channel.
        out_chls (int): Middle channel.
        out_channels (int): Output channel.
    Returns:
        Tuple, tuple of output tensor,(f1,f2,f3).

    Examples:
        YoloBlock(1024, 512, 255).
    """
    def __init__(self, in_channels, out_chls, out_channels):
        super(YoloBlock, self).__init__()
        out_chls_2 = out_chls * 2

        self.conv0 = _conv_bn_relu(in_channels, out_chls, ksize=1)
        self.conv1 = _conv_bn_relu(out_chls, out_chls_2, ksize=3)

        self.conv2 = _conv_bn_relu(out_chls_2, out_chls, ksize=1)
        self.conv3 = _conv_bn_relu(out_chls, out_chls_2, ksize=3)

        self.conv4 = _conv_bn_relu(out_chls_2, out_chls, ksize=1)
```

```
        self.conv5 = _conv_bn_relu(out_chls, out_chls_2, ksize=3)

        self.conv6 = nn.Conv2d(out_chls_2, out_channels, kernel_size=1, stride=1,
has_bias=True)

    def construct(self, x):
        c1 = self.conv0(x)
        c2 = self.conv1(c1)
        c3 = self.conv2(c2)
        c4 = self.conv3(c3)

        c5 = self.conv4(c4)
        c6 = self.conv5(c5)

        out = self.conv6(c6)

        #此处代码实践中有挖空设计，运行代码文件时需补充
        return c5, out


# 定义 YOLOv3 整体网络
class YOLOv3(nn.Cell):
    """
    YOLOv3 Network.

    Note:
        backbone = resnet18.

    Args:
        feature_shape (list): Input image shape, [N,C,H,W].
        backbone_shape (list): resnet18 output channels shape.
        backbone (Cell): Backbone Network.
        out_channel (int): Output channel.

    Returns:
        Tensor, output tensor.

    Examples:
        YOLOv3(feature_shape=[1,3,416,416],
               backbone_shape=[64, 128, 256, 512, 1024]
               backbone=darknet53(),
               out_channel=255).
    """
    def __init__(self, feature_shape, backbone_shape, backbone, out_channel):
        super(YOLOv3, self).__init__()
        self.out_channel = out_channel
        self.net = backbone
```

```
        self.backblock0 = YoloBlock(backbone_shape[-1], out_chls=backbone_shape[-2],
out_channels=out_channel)

        self.conv1 = _conv_bn_relu(in_channel=backbone_shape[-2],
out_channel=backbone_shape[-2]//2, ksize=1)
        self.upsample1 = P.ResizeNearestNeighbor((feature_shape[2]//16, feature_shape[3]//16))
        self.backblock1 = YoloBlock(in_channels=backbone_shape[-2]+backbone_shape[-3],
                                    out_chls=backbone_shape[-3],
                                    out_channels=out_channel)

        self.conv2 = _conv_bn_relu(in_channel=backbone_shape[-3],
out_channel=backbone_shape[-3]//2, ksize=1)
        self.upsample2 = P.ResizeNearestNeighbor((feature_shape[2]//8, feature_shape[3]//8))
        self.backblock2 = YoloBlock(in_channels=backbone_shape[-3]+backbone_shape[-4],
                                    out_chls=backbone_shape[-4],
                                    out_channels=out_channel)
        self.concat = P.Concat(axis=1)

    def construct(self, x):
        # input_shape of x is (batch_size, 3, h, w)
        # feature_map1 is (batch_size, backbone_shape[2], h/8, w/8)
        # feature_map2 is (batch_size, backbone_shape[3], h/16, w/16)
        # feature_map3 is (batch_size, backbone_shape[4], h/32, w/32)
        feature_map1, feature_map2, feature_map3 = self.net(x)
        con1, big_object_output = self.backblock0(feature_map3)

        con1 = self.conv1(con1)
        ups1 = self.upsample1(con1)
        con1 = self.concat((ups1, feature_map2))
        con2, medium_object_output = self.backblock1(con1)

        con2 = self.conv2(con2)
        ups2 = self.upsample2(con2)
        con3 = self.concat((ups2, feature_map1))
        _, small_object_output = self.backblock2(con3)

        return big_object_output, medium_object_output, small_object_output
```

1.4.4.3 定义检测网络

检测网络的目标是从上面的特征中提取有用的框。检测网络的输入为特征提取网络的输出。

检测网络输出介绍：

1) 训练网络返回值为： grid, prediction, box_xy, box_wh。

- grid：为网格；

- prediction：为预测值，但是并非绝对预测值。而是相对值。此预测中心点坐标为相对于其所在网格左上角的偏移坐标；此预测网格宽高值为相对于其对应的锚点的偏移坐标。即：prediction 与网格和锚点对应。
- box_xy：为预测中心点坐标，为 prediction 转换后的绝对坐标；
- box_wh：为预测宽高。为 prediction 转换后的绝对坐标。

2) 测试网络返回值为：box_xy, box_wh, box_confidence, box_probs。

- box_xy：为预测中心点坐标，为绝对坐标；
- box_wh：为预测宽高。为绝对坐标；
- box_confidence：为预测框置信度；
- box_probs：为预测框类别；

下表为检测网络输出维度说明（以小尺度为例）：

名称	维度	描述
grid	(1, 1,11, 20, 1, 1)	网格，这里网格进行了转置[w/32, h/32]
prediction	(32, 11,20, 3, 8)	预测相对结果。[batch_size, h/32, w/32, num_anchors_per_scale, num_attrib]
box_xy	(32, 11,20, 3, 2)	预测中心点绝对坐标。[batch_size, h/32, w/32, num_anchors_per_scale, num_attrib]
box_wh	(32, 11,20, 3, 2)	预测宽高绝对宽高。[batch_size, h/32, w/32, num_anchors_per_scale, num_attrib]
box_confidence	(32, 11,20, 3, 1)	预测绝对置信度。[batch_size, h/32, w/32, num_anchors_per_scale, num_attrib]
box_probs	(32, 11,20, 3, 3)	预测绝对类别。[batch_size, h/32, w/32, num_anchors_per_scale, num_attrib]

表1.4.4.3.1.1.1.1.1

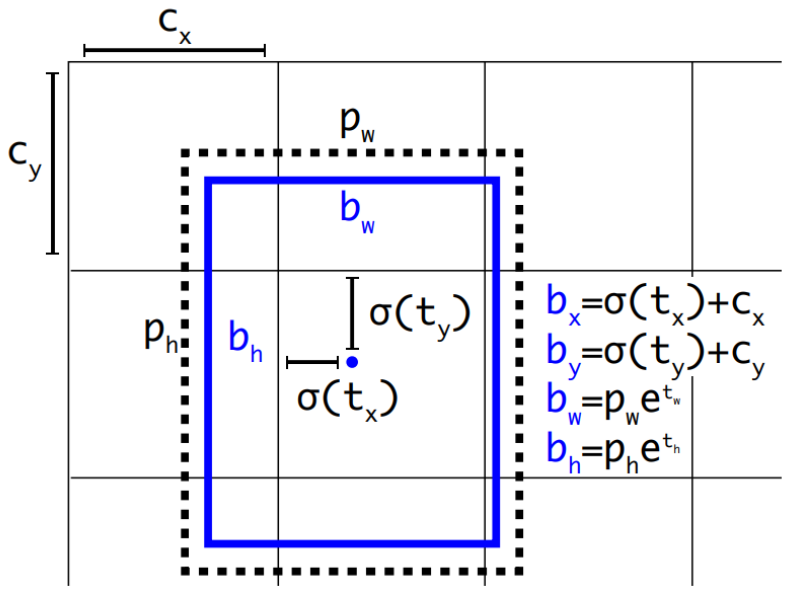


图1.4.4.3.1.1.1.2

我们假设偏移量 prediction 各分量为：t_x 、t_y 、t_w 、t_h 、$t_{confidence}$ 、t_{probs} ，它们分别是：中心点相对于所在网格左上角偏移量 t_x 、t_y ， 宽高相对于锚点偏移量 t_w 、t_h ，置信度和类别为 $t_{confidence}$ 、t_{probs} 。假设预测框绝对值为：b_x 、b_y 、b_w 、b_h 、$b_{confidence}$ 、b_{probs} ，它们分别为：中心点坐标 b_x 、b_y ， 宽高 b_w 、b_h ，置信度和类别为 $b_{confidence}$ 、b_{probs} 。假设网格点为 c_x 、c_y ，假设锚点宽高为 p_w 、p_h 。相对量（或偏移量）和绝对量之间的转换公式如下公式所示：

$$b_x=\sigma(t_x)+c_x$$

$$b_y=\sigma(t_y)+c_y$$

$$b_w=p_w*e^{t_w}$$

$$b_h=p_h*e^{t_h}$$

$$b_{confidence}=\sigma(t_{confidence})$$

$$b_{probs}=\sigma(b_{probs})$$

其中 σ 为 sigmoid 函数，确保偏移量在（0，1）范围内。添加这样的转换主要是因为在训练时如果没有将和压缩到(0,1)区间内的话，模型在训练前期很难收敛。

我们设定的锚点只有宽高，没有中心点，算法默认所有网格都有这些锚点，且中心点为网格左上角坐标。以小尺度为例，共有锚点 11×20×3 个。

计算得到的相对坐标（prediction）每个网格中都有 3 个（小尺度），中心点为每个网格的左上角坐标。

（ c_x 、c_y ）为所有网格的左上角坐标。并非（0，0）点。

```
# 定义检测网络，从上面的特征中提取有用的框
class DetectionBlock(nn.Cell):
    """
    YOLOv3 detection Network. It will finally output the detection result.

    Args:
        scale (str): Character, scale.
        config (Class): YOLOv3 config.

    Returns:
```

```
        Tuple, tuple of output tensor,(f1,f2,f3).

    Examples:
        DetectionBlock(scale='l',stride=32).
    """

    def __init__(self, scale, config):
        super(DetectionBlock, self).__init__()

        self.config = config
        if scale == 's':
            idx = (0, 1, 2)
        elif scale == 'm':
            idx = (3, 4, 5)
        elif scale == 'l':
            idx = (6, 7, 8)
        else:
            raise KeyError("Invalid scale value for DetectionBlock")
        self.anchors = Tensor([self.config.anchor_scales[i] for i in idx], ms.float32)
        self.num_anchors_per_scale = 3
        self.num_attrib = 4 + 1 + self.config.num_classes
        self.ignore_threshold = 0.5
        self.lambda_coord = 1

        self.sigmoid = nn.Sigmoid()
        self.reshape = P.Reshape()
        self.tile = P.Tile()
        self.concat = P.Concat(axis=-1)
        self.input_shape = Tensor(tuple(config.img_shape[::-1]), ms.float32)

    def construct(self, x):
        num_batch = P.Shape()(x)[0]
        grid_size = P.Shape()(x)[2:4]

        # Reshape and transpose the feature to [n, 3, grid_size[0], grid_size[1], num_attrib]
        prediction = P.Reshape()(x, (num_batch,
                                     self.num_anchors_per_scale,
                                     self.num_attrib,
                                     grid_size[0],
                                     grid_size[1]))
        prediction = P.Transpose()(prediction, (0, 3, 4, 1, 2))

        range_x = range(grid_size[1])
        range_y = range(grid_size[0])
        grid_x = P.Cast()(F.tuple_to_array(range_x), ms.float32)
        grid_y = P.Cast()(F.tuple_to_array(range_y), ms.float32)
        # Tensor of shape [grid_size[0], grid_size[1], 1, 1] representing the coordinate of x/y axis
for each grid
```

```
        grid_x = self.tile(self.reshape(grid_x, (1, 1, -1, 1, 1)), (1, grid_size[0], 1, 1, 1))
        grid_y = self.tile(self.reshape(grid_y, (1, -1, 1, 1, 1)), (1, 1, grid_size[1], 1, 1))
        # Shape is [grid_size[0], grid_size[1], 1, 2]
        grid = self.concat((grid_x, grid_y))



        box_xy = prediction[:, :, :, :, :2]
        box_wh = prediction[:, :, :, :, 2:4]
        box_confidence = prediction[:, :, :, :, 4:5]
        box_probs = prediction[:, :, :, :, 5:]

        box_xy = (self.sigmoid(box_xy) + grid) / P.Cast()(F.tuple_to_array((grid_size[1],
grid_size[0])), ms.float32)
        box_wh = P.Exp()(box_wh) * self.anchors / self.input_shape
        box_confidence = self.sigmoid(box_confidence)
        box_probs = self.sigmoid(box_probs)

        if self.training:
            return grid, prediction, box_xy, box_wh
        return box_xy, box_wh, box_confidence, box_probs
```

1.4.4.4 定义 IoU

交并比（Intersection over Union，IoU）：指 Prediction 和 Ground Truth 的交集和并集的比值。

$$IoU=\frac{target \cap prediction}{target \cup prediction}=\frac{intersect_{area}}{box1_{area}+box2_{area}-intersect_{area}}$$

```
# 计算框的 IoU 值
class Iou(nn.Cell):
    """Calculate the iou of boxes."""
    def __init__(self):
        super(Iou, self).__init__()
        self.min = P.Minimum()
        self.max = P.Maximum()

    def construct(self, box1, box2):
        box1_xy = box1[:, :, :, :, :, :2]
        box1_wh = box1[:, :, :, :, :, 2:4]
        box1_mins = box1_xy - box1_wh / F.scalar_to_array(2.0)
        box1_maxs = box1_xy + box1_wh / F.scalar_to_array(2.0)

        box2_xy = box2[:, :, :, :, :, :2]
        box2_wh = box2[:, :, :, :, :, 2:4]
        box2_mins = box2_xy - box2_wh / F.scalar_to_array(2.0)
        box2_maxs = box2_xy + box2_wh / F.scalar_to_array(2.0)
```

```
        intersect_mins = self.max(box1_mins, box2_mins)
        intersect_maxs = self.min(box1_maxs, box2_maxs)
        intersect_wh = self.max(intersect_maxs - intersect_mins, F.scalar_to_array(0.0))

        # 交集
        intersect_area = P.Squeeze(-1)(intersect_wh[:, :, :, :, 0:1]) * \
                         P.Squeeze(-1)(intersect_wh[:, :, :, :, 1:2])
        box1_area = P.Squeeze(-1)(box1_wh[:, :, :, :, 0:1]) * P.Squeeze(-1)(box1_wh[:, :, :, :, :,
1:2])
        box2_area = P.Squeeze(-1)(box2_wh[:, :, :, :, 0:1]) * P.Squeeze(-1)(box2_wh[:, :, :, :, :,
1:2])
        # 并集为(box1_area + box2_area - intersect_area)
        iou = intersect_area / (box1_area + box2_area - intersect_area)
        return iou
```

1.4.4.5 定义 loss 计算

loss 计算主要使用偏移量 prediction 计算，旨在让网络预测值与目标值的偏移最小。

输入为预测值和真实值，输出为 loss 值。

输入：grid，prediction，pred_xy（即 box_xy），pred_wh(即 box_wh)，y_true，gt_box；输出为 loss。

YOLOv3 的 loss 由多部分相加组成，包含：

- 中心点 loss—— xy_loss（交叉熵）
- 宽高 loss—— wh_loss（L2 损失）
- 置信度 loss—— confidence_loss（真阳性+真阴性）
- 分类 loss—— class_loss（交叉熵）

我们假设偏移量 prediction 各分量为：t_xy、t_wh、t_confidence、t_probs。假设真实值 bbox 各分量为：true_xy、true_wh、true_confidence、true_probs。计算方式如下所示：

$$xy_{loss}=true_{confidence}\times\left(2-true_w\times true_h\right)\times cross_{entropy}\left(t_{xy},true_{xy}\times grid_{shape}-grid\right)$$

$$¿\,t_{wh}-\left(true_{wh}\times grid_{shape}-grid\right)\vee¿^2$$

$$wh_{loss}=true_{confidence}\times\left(2-true_w\times true_h\right)\times\frac{1}{2}¿$$

$$cross_{con}=cross_{entropy}\left(t_{confidence},true_{confidence}\right)$$

$$confidence_{loss}=true_{confidence}\times cross_{con}+\left(1-true_{confidence}\right)\times cross_{con}\times ignore_{mas}k$$

$$class_{loss}=object_{mask}\times cross_{entropy}\left(t_{probs},true_{probs}\right)$$

其中公式 xy_loss、wh_loss 中的 (2−true_w×true_h) 是为了弱化边界框尺寸对损失值的影响，该值为 1-2；

ignore_mask 为求得的忽略框 mask，阈值为 config.py 文件中的 ignore_threshold（0.5）。

```
# 损失函数
class YoloLossBlock(nn.Cell):
    """
    YOLOv3 Loss block cell. It will finally output loss of the scale.

    Args:
        scale (str): Three scale here, 's', 'm' and 'l'.
        config (Class): The default config of YOLOv3.

    Returns:
        Tensor, loss of the scale.

    Examples:
        YoloLossBlock('l', ConfigYOLOV3ResNet18()).
    """

    def __init__(self, scale, config):
        super(YoloLossBlock, self).__init__()
        self.config = config
        if scale == 's':
            idx = (0, 1, 2)
        elif scale == 'm':
            idx = (3, 4, 5)
        elif scale == 'l':
            idx = (6, 7, 8)
        else:
            raise KeyError("Invalid scale value for DetectionBlock")
        self.anchors = Tensor([self.config.anchor_scales[i] for i in idx], ms.float32)
        self.ignore_threshold = Tensor(self.config.ignore_threshold, ms.float32)
        self.concat = P.Concat(axis=-1)
        self.iou = Iou()
        self.cross_entropy = P.SigmoidCrossEntropyWithLogits()
        self.reduce_sum = P.ReduceSum()
        self.reduce_max = P.ReduceMax(keep_dims=False)
        self.input_shape = Tensor(tuple(config.img_shape[::-1]), ms.float32)

    def construct(self, grid, prediction, pred_xy, pred_wh, y_true, gt_box):

        object_mask = y_true[:, :, :, :, 4:5]
        class_probs = y_true[:, :, :, :, 5:]

        grid_shape = P.Shape()(prediction)[1:3]
        grid_shape = P.Cast()(F.tuple_to_array(grid_shape[::-1]), ms.float32)
```

```
        pred_boxes = self.concat((pred_xy, pred_wh))
        true_xy = y_true[:, :, :, :, :2] * grid_shape - grid
        true_wh = y_true[:, :, :, :, 2:4]
        true_wh = P.Select()(P.Equal()(true_wh, 0.0),
                             P.Fill()(P.DType()(true_wh), P.Shape()(true_wh), 1.0),
                             true_wh)
        true_wh = P.Log()(true_wh / self.anchors * self.input_shape)
        box_loss_scale = 2 - y_true[:, :, :, :, 2:3] * y_true[:, :, :, :, 3:4]

        gt_shape = P.Shape()(gt_box)
        gt_box = P.Reshape()(gt_box, (gt_shape[0], 1, 1, 1, gt_shape[1], gt_shape[2]))

        iou = self.iou(P.ExpandDims()(pred_boxes, -2), gt_box) # [batch, grid[0], grid[1],
num_anchor, num_gt]
        # 计算 pred_boxes（预测值，非偏移）和 gt_box 的 IoU 值
        best_iou = self.reduce_max(iou, -1) # [batch, grid[0], grid[1], num_anchor]
        # 比较这些 IoU 值与 ignore_threshold 值大小，结果为 ignore_mask
        ignore_mask = best_iou < self.ignore_threshold
        ignore_mask = P.Cast()(ignore_mask, ms.float32)
        ignore_mask = P.ExpandDims()(ignore_mask, -1)
        # 梯度不进行更新
        ignore_mask = F.stop_gradient(ignore_mask)

        xy_loss = object_mask * box_loss_scale * self.cross_entropy(prediction[:, :, :, :, :2],
true_xy)
        wh_loss = object_mask * box_loss_scale * 0.5 * P.Square()(true_wh - prediction[:, :, :, :,
2:4])
        confidence_loss = self.cross_entropy(prediction[:, :, :, :, 4:5], object_mask)
        confidence_loss = object_mask * confidence_loss + (1 - object_mask) * confidence_loss *
ignore_mask
        class_loss = object_mask * self.cross_entropy(prediction[:, :, :, :, 5:], class_probs)

        # 平滑损失
        xy_loss = self.reduce_sum(xy_loss, ())
        wh_loss = self.reduce_sum(wh_loss, ())
        confidence_loss = self.reduce_sum(confidence_loss, ())
        class_loss = self.reduce_sum(class_loss, ())

        loss = xy_loss + wh_loss + confidence_loss + class_loss
        return loss / P.Shape()(prediction)[0]

# 基于 ResNet18 的 YOLOv3 网络
class yolov3_resnet18(nn.Cell):
    """
    ResNet based YOLOv3 network.

    Args:
        config (Class): YOLOv3 config.
```

```
    Returns:
        Cell, cell instance of ResNet based YOLOv3 neural network.

    Examples:
        yolov3_resnet18(80, [1,3,416,416]).
    """

    def __init__(self, config):
        super(yolov3_resnet18, self).__init__()
        self.config = config

        # YOLOv3 network
        self.feature_map = YOLOv3(feature_shape=self.config.feature_shape,
                                  backbone=ResNet(BasicBlock,
                                                  self.config.backbone_layers,
                                                  self.config.backbone_input_shape,
                                                  self.config.backbone_shape,
                                                  self.config.backbone_stride,
                                                  num_classes=None),
                                  backbone_shape=self.config.backbone_shape,
                                  out_channel=self.config.out_channel)

        # prediction on the default anchor boxes
        self.detect_1 = DetectionBlock('l', self.config)
        self.detect_2 = DetectionBlock('m', self.config)
        self.detect_3 = DetectionBlock('s', self.config)

    def construct(self, x):
        big_object_output, medium_object_output, small_object_output = self.feature_map(x)
        output_big = self.detect_1(big_object_output)
        output_me = self.detect_2(medium_object_output)
        output_small = self.detect_3(small_object_output)

        return output_big, output_me, output_small


# 训练网络的损失计算
class YoloWithLossCell(nn.Cell):
    """
    Provide YOLOv3 training loss through network.

    Args:
        network (Cell): The training network.
        config (Class): YOLOv3 config.

    Returns:
```

```
        Tensor, the loss of the network.
    """
    def __init__(self, network, config):
        super(YoloWithLossCell, self).__init__()
        self.yolo_network = network
        self.config = config
        self.loss_big = YoloLossBlock('l', self.config)
        self.loss_me = YoloLossBlock('m', self.config)
        self.loss_small = YoloLossBlock('s', self.config)

    def construct(self, x, y_true_0, y_true_1, y_true_2, gt_0, gt_1, gt_2):
        yolo_out = self.yolo_network(x)
        loss_l = self.loss_big(yolo_out[0][0], yolo_out[0][1], yolo_out[0][2], yolo_out[0][3],
y_true_0, gt_0)
        loss_m = self.loss_me(yolo_out[1][0], yolo_out[1][1], yolo_out[1][2], yolo_out[1][3],
y_true_1, gt_1)
        loss_s = self.loss_small(yolo_out[2][0], yolo_out[2][1], yolo_out[2][2], yolo_out[2][3],
y_true_2, gt_2)
        return loss_l + loss_m + loss_s


# YOLOv3 训练网络封装
class TrainingWrapper(nn.Cell):
    """
    Encapsulation class of YOLOv3 network training.

    Append an optimizer to the training network after that the construct
    function can be called to create the backward graph.

    Args:
        network (Cell): The training network. Note that loss function should have been added.
        optimizer (Optimizer): Optimizer for updating the weights.
        sens (Number): The adjust parameter. Default: 1.0.
    """
    def __init__(self, network, optimizer, sens=1.0):
        super(TrainingWrapper, self).__init__(auto_prefix=False)
        self.network = network
        self.network.set_grad()
        self.weights = ms.ParameterTuple(network.trainable_params())
        self.optimizer = optimizer
        self.grad = C.GradOperation(get_by_list=True, sens_param=True)
        self.sens = sens
        self.reducer_flag = False
        self.grad_reducer = None
        self.parallel_mode = context.get_auto_parallel_context("parallel_mode")
        if self.parallel_mode in [ParallelMode.DATA_PARALLEL, ParallelMode.HYBRID_PARALLEL]:
            self.reducer_flag = True
        if self.reducer_flag:
```

```
        mean = context.get_auto_parallel_context("gradients_mean")
        if auto_parallel_context().get_device_num_is_set():
            degree = context.get_auto_parallel_context("device_num")
        else:
            degree = get_group_size()
        self.grad_reducer = nn.DistributedGradReducer(optimizer.parameters, mean, degree)

    def construct(self, *args):
        weights = self.weights
        loss = self.network(*args)
        sens = P.Fill()(P.DType()(loss), P.Shape()(loss), self.sens)
        grads = self.grad(self.network, weights)(*args, sens)
        if self.reducer_flag:
            # apply grad reducer on grads
            grads = self.grad_reducer(grads)
        return F.depend(loss, self.optimizer(grads))
```

1.4.4.6 定义 YOLOv3 验证网络

框反映射和分数计算（class YoloBoxScores）：

在数据预处理部分，为了统一尺度我们将 image 和 box 都映射到尺度为[352,640]的空间中。为了得到原始图片中框的大小，需要对 yolov3_resnet18 求得的框进行反映射。得到结果同原始数据格式处理中 annotation。class YoloBoxScores 的输出结果为：box 和 boxes_scores。

box、boxes_scores 介绍如下所示（以小尺度为例）：

名称	维度	描述
box	(32, 11×20×311×20×3，4)	预测框，共 11×20×311×20×3 个,每个框由四个标签，[xmin,ymin,xmax,ymax]
boxes_scores	(32, 11×20×311×20×3，3)	预测框分数（3 个类别），共 11×20×311×20×3 个框，每个框由 3 个分数。

表1.4.4.6.1.1.1.1.1

上表中的 32 代表 batch_size；

此 YOLOv3 网络共可以得到 13860 个框，是三个尺度框数的总和：

$$13860=11\times20\times3+22\times40\times3+44\times80\times3$$

boxes_scores 为框的分数和类别分数的乘积，即：

$$boxes_{scores}=box_{confidence}*box_{probs}$$

```
# 计算原图片的框大小和每个框的得分
class YoloBoxScores(nn.Cell):
    """
    Calculate the boxes of the original picture size and the score of each box.
```

```
    Args:
        config (Class): YOLOv3 config.

    Returns:
        Tensor, the boxes of the original picture size.
        Tensor, the score of each box.
    """
    def __init__(self, config):
        super(YoloBoxScores, self).__init__()
        self.input_shape = Tensor(np.array(config.img_shape), ms.float32)
        self.num_classes = config.num_classes

    def construct(self, box_xy, box_wh, box_confidence, box_probs, image_shape):
        batch_size = F.shape(box_xy)[0]
        x = box_xy[:, :, :, 0:1]
        y = box_xy[:, :, :, 1:2]
        box_yx = P.Concat(-1)((y, x))
        w = box_wh[:, :, :, 0:1]
        h = box_wh[:, :, :, 1:2]
        box_hw = P.Concat(-1)((h, w))

        new_shape = P.Round()(image_shape * P.ReduceMin()(self.input_shape / image_shape))
        offset = (self.input_shape - new_shape) / 2.0 / self.input_shape
        scale = self.input_shape / new_shape
        box_yx = (box_yx - offset) * scale
        box_hw = box_hw * scale

        box_min = box_yx - box_hw / 2.0
        box_max = box_yx + box_hw / 2.0
        boxes = P.Concat(-1)((box_min[:, :, :, 0:1],
                              box_min[:, :, :, 1:2],
                              box_max[:, :, :, 0:1],
                              box_max[:, :, :, 1:2]))
        image_scale = P.Tile()(image_shape, (1, 2))
        boxes = boxes * image_scale
        boxes = F.reshape(boxes, (batch_size, -1, 4))
        boxes_scores = box_confidence * box_probs
        boxes_scores = F.reshape(boxes_scores, (batch_size, -1, self.num_classes))
        return boxes, boxes_scores


# 封装 YOLOv3 的验证网络
class YoloWithEval(nn.Cell):
    """
    Encapsulation class of YOLOv3 evaluation.

    Args:
```

```
        network (Cell): The training network. Note that loss function and optimizer must not be
added.
        config (Class): YOLOv3 config.

    Returns:
        Tensor, the boxes of the original picture size.
        Tensor, the score of each box.
        Tensor, the original picture size.
    """
    def __init__(self, network, config):
        super(YoloWithEval, self).__init__()
        self.yolo_network = network
        self.box_score_0 = YoloBoxScores(config)
        self.box_score_1 = YoloBoxScores(config)
        self.box_score_2 = YoloBoxScores(config)

    def construct(self, x, image_shape):
        yolo_output = self.yolo_network(x)
        boxes_0, boxes_scores_0 = self.box_score_0(*yolo_output[0], image_shape)
        boxes_1, boxes_scores_1 = self.box_score_1(*yolo_output[1], image_shape)
        boxes_2, boxes_scores_2 = self.box_score_2(*yolo_output[2], image_shape)
        boxes = P.Concat(1)((boxes_0, boxes_1, boxes_2))
        boxes_scores = P.Concat(1)((boxes_scores_0, boxes_scores_1, boxes_scores_2))
        return boxes, boxes_scores, image_shape
```

1.4.5 评价指标定义

评价指标定义在 code/src/utils.py，无需执行，“iou 计算”部分代码设计了挖空需补全。

非极大值抑制 NMS 算法：

由于滑动窗口，同一个 class 可能有好几个框(每一个框都带有一个分类器得分)，我们的目的就是要去除冗余的检测框，保留最好的一个。于是我们就要用到非极大值抑制，来抑制那些冗余的框： 抑制的过程是一个迭代-遍历-消除的过程。

- 将 person 类别所有框的得分排序，选中最高分及其对应的框 A：
- 遍历其余的框，如果和当前最高分框 A 的重叠面积(IOU)大于一定阈值，我们就将框删除。
- 从剩下的 person 类别框中继续选一个得分最高的（非 A，A 已经确定），重复上述过程，指导找到所有满足与之的 person 类别框。
- 重复上述过程，找到所有满足条件的 facce 类别框和 mask 类别框。

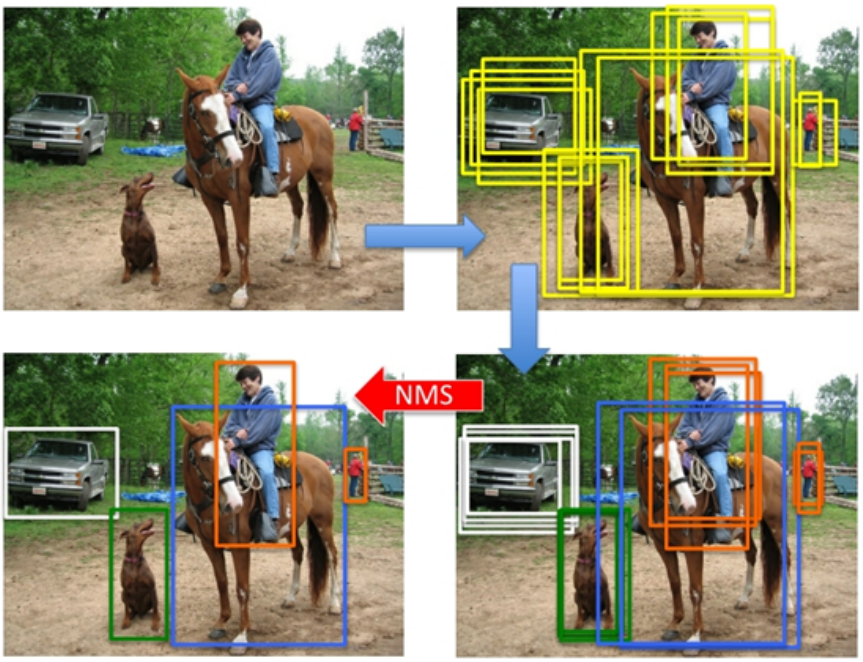


图1.4.5.1.1.1.1.1

```
import numpy as np
from src.config import ConfigYOLOV3ResNet18

#计算预测框和真实值的 IoU 值。
def calc_iou(bbox_pred, bbox_ground):
    """Calculate iou of predicted bbox and ground truth."""
    x1 = bbox_pred[0]
    y1 = bbox_pred[1]
    width1 = bbox_pred[2] - bbox_pred[0]
    height1 = bbox_pred[3] - bbox_pred[1]

    x2 = bbox_ground[0]
    y2 = bbox_ground[1]
    width2 = bbox_ground[2] - bbox_ground[0]
    height2 = bbox_ground[3] - bbox_ground[1]

    endx = max(x1 + width1, x2 + width2)
    startx = min(x1, x2)
    width = width1 + width2 - (endx - startx)

    endy = max(y1 + height1, y2 + height2)
    starty = min(y1, y2)
    height = height1 + height2 - (endy - starty)

    if width <= 0 or height <= 0:
        iou = 0
```

```
        else:

        #此处 iou 计算，在代码实践中有挖空设计，需自行补充
        area = width * height
        area1 = width1 * height1
        area2 = width2 * height2
        iou = area * 1. / (area1 + area2 - area)

    return iou


# 将极大值抑制应用于框
def apply_nms(all_boxes, all_scores, thres, max_boxes):
    """Apply NMS to bboxes."""
    x1 = all_boxes[:, 0]
    y1 = all_boxes[:, 1]
    x2 = all_boxes[:, 2]
    y2 = all_boxes[:, 3]
    areas = (x2 - x1 + 1) * (y2 - y1 + 1)

    order = all_scores.argsort()[::-1]
    keep = []

    while order.size > 0:
        i = order[0]
        keep.append(i)

        if len(keep) >= max_boxes:
            break

        xx1 = np.maximum(x1[i], x1[order[1:]])
        yy1 = np.maximum(y1[i], y1[order[1:]])
        xx2 = np.minimum(x2[i], x2[order[1:]])
        yy2 = np.minimum(y2[i], y2[order[1:]])

        w = np.maximum(0.0, xx2 - xx1 + 1)
        h = np.maximum(0.0, yy2 - yy1 + 1)
        inter = w * h

        ovr = inter / (areas[i] + areas[order[1:]] - inter)

        inds = np.where(ovr <= thres)[0]

        order = order[inds + 1]
    return keep
```

```
# 计算预测框的精度和召回率
def metrics(pred_data):
    """Calculate precision and recall of predicted bboxes."""
    config = ConfigYOLOV3ResNet18()
    num_classes = config.num_classes
    count_corrects = [1e-6 for _ in range(num_classes)]
    count_grounds = [1e-6 for _ in range(num_classes)]
    count_preds = [1e-6 for _ in range(num_classes)]

    for i, sample in enumerate(pred_data):
        gt_anno = sample["annotation"]
        box_scores = sample['box_scores']
        boxes = sample['boxes']
        mask = box_scores >= config.obj_threshold
        boxes_ = []
        scores_ = []
        classes_ = []
        max_boxes = config.nms_max_num
        for c in range(num_classes):
            class_boxes = np.reshape(boxes, [-1, 4])[np.reshape(mask[:, c], [-1])]
            class_box_scores = np.reshape(box_scores[:, c], [-1])[np.reshape(mask[:, c], [-1])]
            nms_index = apply_nms(class_boxes, class_box_scores, config.nms_threshold,
max_boxes)
            class_boxes = class_boxes[nms_index]
            class_box_scores = class_box_scores[nms_index]
            classes = np.ones_like(class_box_scores, 'int32') * c
            boxes_.append(class_boxes)
            scores_.append(class_box_scores)
            classes_.append(classes)

        boxes = np.concatenate(boxes_, axis=0)
        classes = np.concatenate(classes_, axis=0)


        # metric
        count_correct = [1e-6 for _ in range(num_classes)]
        count_ground = [1e-6 for _ in range(num_classes)]
        count_pred = [1e-6 for _ in range(num_classes)]

        for anno in gt_anno:
            count_ground[anno[4]] += 1

        for box_index, box in enumerate(boxes):
            bbox_pred = [box[1], box[0], box[3], box[2]]
            count_pred[classes[box_index]] += 1

            for anno in gt_anno:
                class_ground = anno[4]
```

```
            if classes[box_index] == class_ground:
                iou = calc_iou(bbox_pred, anno)
                if iou >= 0.5:
                    count_correct[class_ground] += 1
                    break

        count_corrects = [count_corrects[i] + count_correct[i] for i in range(num_classes)]
        count_preds = [count_preds[i] + count_pred[i] for i in range(num_classes)]
        count_grounds = [count_grounds[i] + count_ground[i] for i in range(num_classes)]

    precision = np.array([count_corrects[ix] / count_preds[ix] for ix in range(num_classes)])
    recall = np.array([count_corrects[ix] / count_grounds[ix] for ix in range(num_classes)])
    return precision, recall
```

1.4.6 超参数定义

超参数定义文件在 code/src/config.py，无需执行。

这里通过定义一个类来定义所有超参数。

```
"""Config parameters for YOLOv3 models."""


class ConfigYOLOV3ResNet18:
    """
    Config parameters for YOLOv3.

    Examples:
        ConfigYoloV3ResNet18.
    """
    img_shape = [352, 640]  # 本实验将所有图片都统一为[352,640]进行训练和推理
    feature_shape = [32, 3, 352, 640]  # 网络输入图片[batch_size,channel,weight,height]
    num_classes = 3  # 类别：person, face, mask
    nms_max_num = 50  # 一张图片最多可以保存 50 个框
    _NUM_BOXES = 50  # 最多框，50

    backbone_input_shape = [64, 64, 128, 256]  # 主干网络的输入形状
    backbone_shape = [64, 128, 256, 512]  # 主干网络的输出形状

    backbone_layers = [2, 2, 2, 2]  # 主干网络的层数
    backbone_stride = [1, 2, 2, 2]  # 主干网络的步长

    ignore_threshold = 0.5  # 忽略框的阈值
    obj_threshold = 0.3
    nms_threshold = 0.4  # NMS 去除冗余检测框的阈值

     # 锚点，根据标签框的大小来设定
    anchor_scales = [(5,3),(10, 13), (16, 30),(33, 23),(30, 61),(62, 45),(59, 119),(116, 90),(156,
198)]
```

```
    out_channel = int(len(anchor_scales) / 3 * (num_classes + 5))  # 输出通道
```

1.4.7 项目执行

目标检测项目的执行文件为 code/main.ipynb，需要执行。

步骤 1　环境导入

```
import os
import argparse
import ast
from easydict import EasyDict as edict
import shutil

#此处在代码实践环节设置有代码挖空
import numpy as np
import mindspore.nn as nn
from mindspore import context, Tensor

from mindspore.communication.management import init
from mindspore.train.callback import CheckpointConfig, ModelCheckpoint, LossMonitor,
TimeMonitor

#此处在代码实践环节设置有代码挖空
from mindspore.train import Model

from mindspore.context import ParallelMode
from mindspore.train.serialization import load_checkpoint, load_param_into_net
from mindspore.common.initializer import initializer
from mindspore.common import set_seed

import sys
sys.path.insert(0,'./yolov3/yolov3_resnet18/')     #yours code path
#sys.path.insert(0,'./yolov3/code/')
from src.yolov3 import yolov3_resnet18, YoloWithLossCell, TrainingWrapper
from src.dataset import create_yolo_dataset, data_to_mindrecord_byte_image
from src.config import ConfigYOLOV3ResNet18

import moxing as mox

set_seed(1)
```

步骤 2　训练网络定义

```
# 定义学习率
def get_lr(learning_rate, start_step, global_step, decay_step, decay_rate, steps=False):
    """Set learning rate."""
    lr_each_step = []
    for i in range(global_step):
```

```
        if steps:
            lr_each_step.append(learning_rate * (decay_rate ** (i // decay_step)))
        else:
            lr_each_step.append(learning_rate * (decay_rate ** (i / decay_step)))
    lr_each_step = np.array(lr_each_step).astype(np.float32)
    lr_each_step = lr_each_step[start_step:]
    return lr_each_step


# 定义网络初始化参数
def init_net_param(network, init_value='ones'):
    """Init the parameters in network."""
    params = network.trainable_params()
    for p in params:
        if isinstance(p.data, Tensor) and 'beta' not in p.name and 'gamma' not in p.name and
'bias' not in p.name:
            p.set_data(initializer(init_value, p.data.shape, p.data.dtype))

# 定义训练网络
def main(args_opt):
    context.set_context(mode=context.GRAPH_MODE, device_target="Ascend")
    if args_opt.distribute:
        device_num = args_opt.device_num
        context.reset_auto_parallel_context()
        context.set_auto_parallel_context(parallel_mode=ParallelMode.DATA_PARALLEL,
gradients_mean=True,
                                          device_num=device_num)
        init()
        rank = args_opt.device_id % device_num
    else:
        rank = 0
        device_num = 1

    loss_scale = float(args_opt.loss_scale)

    # When create MindDataset, using the fitst mindrecord file, such as yolo.mindrecord0.
    dataset = create_yolo_dataset(args_opt.mindrecord_file,
                                  batch_size=args_opt.batch_size, device_num=device_num, rank=rank)
    dataset_size = dataset.get_dataset_size()
    print('The step size: ', dataset_size)
    print("Create dataset done!")

    net = yolov3_resnet18(ConfigYOLOV3ResNet18())
    net = YoloWithLossCell(net, ConfigYOLOV3ResNet18())
    init_net_param(net, "XavierUniform")

    # checkpoint
```

```
    ckpt_config = CheckpointConfig(save_checkpoint_steps=dataset_size *
args_opt.save_checkpoint_epochs,
                                   keep_checkpoint_max=args_opt.keep_checkpoint_max)
    ckpoint_cb = ModelCheckpoint(prefix="yolov3", directory=cfg.ckpt_dir, config=ckpt_config)

    if args_opt.pre_trained:
        if args_opt.pre_trained_epoch_size <= 0:
            raise KeyError("pre_trained_epoch_size must be greater than 0.")
        param_dict = load_checkpoint(args_opt.pre_trained)
        load_param_into_net(net, param_dict)
    total_epoch_size = 60
    if args_opt.distribute:
        total_epoch_size = 160
    lr = Tensor(get_lr(learning_rate=args_opt.lr, start_step=args_opt.pre_trained_epoch_size *
dataset_size,
                       global_step=total_epoch_size * dataset_size,
                       decay_step=1000, decay_rate=0.95, steps=True))
    opt = nn.Adam(filter(lambda x: x.requires_grad, net.get_parameters()), lr,
loss_scale=loss_scale)
    net = TrainingWrapper(net, opt, loss_scale)

    callback = [LossMonitor(10*dataset_size), ckpoint_cb]
    model = Model(net)
    dataset_sink_mode = cfg.dataset_sink_mode
    print("Start train YOLOv3, the first epoch will be slower because of the graph compilation.")
    model.train(args_opt.epoch_size, dataset, callbacks=callback,
dataset_sink_mode=dataset_sink_mode)
```

步骤 3　超参数定义

```
cfg = edict({
    "distribute": False,
#     "device_id": 0,
#     "device_num": 1,
    "dataset_sink_mode": True,

    "lr": 0.001,
    "epoch_size": 60,
    "batch_size": 32,
    "loss_scale" : 1024,

    "pre_trained": None,
    "pre_trained_epoch_size":0,

    "ckpt_dir": "./ckpt",
    "save_checkpoint_epochs" :1,
    'keep_checkpoint_max': 1,
})
```

步骤 4　启动训练

```
if os.path.exists(cfg.ckpt_dir):
    shutil.rmtree(cfg.ckpt_dir)
data_path = '../data/'

mindrecord_dir_train = os.path.join(data_path,'mindrecord/train')

print("Start create dataset!")
# It will generate mindrecord file in args_opt.mindrecord_dir,and the file name is
yolo.mindrecord.
prefix = "yolo.mindrecord"
cfg.mindrecord_file = os.path.join(mindrecord_dir_train, prefix)
if os.path.exists(mindrecord_dir_train):
    shutil.rmtree(mindrecord_dir_train)

image_dir = os.path.join(data_path, "train")
if os.path.exists(mindrecord_dir_train) and os.listdir(mindrecord_dir_train):
    print('The mindrecord file had exists!')
else:
    image_dir = os.path.join(data_path, "train")
    if not os.path.exists(mindrecord_dir_train):
        os.makedirs(mindrecord_dir_train)
    print("Create Mindrecord.")
    data_to_mindrecord_byte_image(image_dir, mindrecord_dir_train, prefix, 1)
    print("Create Mindrecord Done, at {}".format(mindrecord_dir_train))
    # if you need use mindrecord file next time, you can save them to yours obs.
    #mox.file.copy_parallel(src_url=args_opt.mindrecord_dir_train,
dst_url=os.path.join(cfg.data_url,'mindspore/train')

# 执行训练
main(cfg)
```

输出（WARNING 不用管）：

```
Start create dataset!
Create Mindrecord.
Create Mindrecord Done, at ./data/mindrecord/train
The epoch size:  15
Create dataset done!
[WARNING] ME(1448:281473694427440,MainProcess):2021-03-22-05:57:26.559.631 [mindspore/ops/operations/math_ops.py:171] WARN_DEPRECATED: The u
sage of TensorAdd is deprecated. Please use Add.
[WARNING] ME(1448:281473694427440,MainProcess):2021-03-22-05:57:26.586.825 [mindspore/ops/operations/math_ops.py:171] WARN_DEPRECATED: The u
sage of TensorAdd is deprecated. Please use Add.
[WARNING] ME(1448:281473694427440,MainProcess):2021-03-22-05:57:26.637.802 [mindspore/ops/operations/math_ops.py:171] WARN_DEPRECATED: The u
sage of TensorAdd is deprecated. Please use Add.
[WARNING] ME(1448:281473694427440,MainProcess):2021-03-22-05:57:26.692.018 [mindspore/ops/operations/math_ops.py:171] WARN_DEPRECATED: The u
sage of TensorAdd is deprecated. Please use Add.
[WARNING] ME(1448:281473694427440,MainProcess):2021-03-22-05:57:26.845.530 [mindspore/ops/operations/math_ops.py:171] WARN_DEPRECATED: The u
sage of TensorAdd is deprecated. Please use Add.
[WARNING] ME(1448:281473694427440,MainProcess):2021-03-22-05:57:27.369.14 [mindspore/ops/operations/math_ops.py:171] WARN_DEPRECATED: The us
age of TensorAdd is deprecated. Please use Add.
[WARNING] ME(1448:281473694427440,MainProcess):2021-03-22-05:57:27.632.598 [mindspore/ops/operations/math_ops.py:171] WARN_DEPRECATED: The u
sage of TensorAdd is deprecated. Please use Add.
[WARNING] ME(1448:281473694427440,MainProcess):2021-03-22-05:57:28.381.564 [mindspore/ops/operations/math_ops.py:171] WARN_DEPRECATED: The u
sage of TensorAdd is deprecated. Please use Add.

Start train YOLOv3, the first epoch will be slower because of the graph compilation.
epoch: 10 step: 15, loss is 223.67534
epoch: 20 step: 15, loss is 148.1925
epoch: 30 step: 15, loss is 101.994064
epoch: 40 step: 15, loss is 92.660255
epoch: 50 step: 15, loss is 84.48035
epoch: 60 step: 15, loss is 61.228745
```

步骤 5 测试网络定义

YOLOv3 推理需要从前面得到的 13860 个框中得到我们需要的有用框（见 main.ipynb 文件的 tobox 函数），其中从众多框中选择有效框算法为 nms 算法（见 utils.py 文件中的 apply_nm 函数）。

```
import os
import argparse
import time
from easydict import EasyDict as edict

import matplotlib.pyplot as plt
from PIL import Image
import PIL
import numpy as np

import sys
#sys.path.insert(0,'./yolov3/code/')
sys.path.insert(0,'./yolov3/yolov3_resnet18/')                # yours code path
import moxing as mox
from mindspore import context, Tensor
from mindspore.train.serialization import load_checkpoint, load_param_into_net
from src.yolov3 import yolov3_resnet18, YoloWithEval
from src.dataset import create_yolo_dataset, data_to_mindrecord_byte_image
from src.config import ConfigYOLOV3ResNet18
from src.utils import metrics


def apply_nms(all_boxes, all_scores, thres, max_boxes):
    """Apply NMS to bboxes."""
    x1 = all_boxes[:, 0]
    y1 = all_boxes[:, 1]
    x2 = all_boxes[:, 2]
    y2 = all_boxes[:, 3]
    areas = (x2 - x1 + 1) * (y2 - y1 + 1)

    order = all_scores.argsort()[::-1]
    keep = []

    while order.size > 0:
        i = order[0]
        keep.append(i)

        if len(keep) >= max_boxes:
            break
```

```
        xx1 = np.maximum(x1[i], x1[order[1:]])
        yy1 = np.maximum(y1[i], y1[order[1:]])
        xx2 = np.minimum(x2[i], x2[order[1:]])
        yy2 = np.minimum(y2[i], y2[order[1:]])

        w = np.maximum(0.0, xx2 - xx1 + 1)
        h = np.maximum(0.0, yy2 - yy1 + 1)
        inter = w * h

        ovr = inter / (areas[i] + areas[order[1:]] - inter)

        inds = np.where(ovr <= thres)[0]

        order = order[inds + 1]
    return keep

def tobox(boxes, box_scores):
    """Calculate precision and recall of predicted bboxes."""
    config = ConfigYOLOV3ResNet18()
    num_classes = config.num_classes
    mask = box_scores >= config.obj_threshold
    boxes_ = []
    scores_ = []
    classes_ = []
    max_boxes = config.nms_max_num
    for c in range(num_classes):
        class_boxes = np.reshape(boxes, [-1, 4])[np.reshape(mask[:, c], [-1])]
        class_box_scores = np.reshape(box_scores[:, c], [-1])[np.reshape(mask[:, c], [-1])]
        nms_index = apply_nms(class_boxes, class_box_scores, config.nms_threshold,
max_boxes)
        #nms_index = apply_nms(class_boxes, class_box_scores, 0.5, max_boxes)
        class_boxes = class_boxes[nms_index]
        class_box_scores = class_box_scores[nms_index]
        classes = np.ones_like(class_box_scores, 'int32') * c
        boxes_.append(class_boxes)
        scores_.append(class_box_scores)
        classes_.append(classes)

    boxes = np.concatenate(boxes_, axis=0)
    classes = np.concatenate(classes_, axis=0)
    scores = np.concatenate(scores_, axis=0)

    return boxes, classes, scores

def yolo_eval(cfg):
    """Yolov3 evaluation."""
    ds = create_yolo_dataset(cfg.mindrecord_file, batch_size=1, is_training=False)
    config = ConfigYOLOV3ResNet18()
```

```
    net = yolov3_resnet18(config)
    eval_net = YoloWithEval(net, config)
    print("Load Checkpoint!")
    param_dict = load_checkpoint(cfg.ckpt_path)
    load_param_into_net(net, param_dict)

    eval_net.set_train(False)
    i = 1.
    total = ds.get_dataset_size()
    start = time.time()
    pred_data = []
    print("\n========================================\n")
    print("total images num: ", total)
    print("Processing, please wait a moment.")

    num_class={0:'person', 1: 'face', 2:'mask'}
    for data in ds.create_dict_iterator(output_numpy=True):
        img_np = data['image']
        image_shape = data['image_shape']
        #print(image_shape)
        annotation = data['annotation']
        image_file = data['file']
        image_file = image_file.tostring().decode('ascii')

        eval_net.set_train(False)
        output = eval_net(Tensor(img_np), Tensor(image_shape))
        for batch_idx in range(img_np.shape[0]):
            boxes = output[0].asnumpy()[batch_idx]
            box_scores = output[1].asnumpy()[batch_idx]
            image = img_np[batch_idx,...]
            boxes, classes, scores =tobox(boxes, box_scores)
            #print(classes)
            #print(scores)
            fig = plt.figure()  #相当于创建画板
            ax = fig.add_subplot(1,1,1)  #创建子图，相当于在画板中添加一个画纸，当然可创建多个画纸，具体由其中参数而定
            image_path = os.path.join(cfg.image_dir, image_file)
            f = Image.open(image_path)
            img_np = np.asarray(f ,dtype=np.float32）#H，W，C格式
            ax.imshow(img_np.astype(np.uint8)）#当前画纸中画一个图片

            for box_index in range(boxes.shape[0]):
                ymin=boxes[box_index][0]
                xmin=boxes[box_index][1]
                ymax=boxes[box_index][2]
                xmax=boxes[box_index][3]
                #print(xmin,ymin,xmax,ymax)
                #添加方框，(xmin,ymin)表示左顶点坐标，(xmax-xmin),(ymax-ymin)表示方框长宽
```

```
            ax.add_patch(plt.Rectangle((xmin,ymin),(xmax-xmin),(ymax-
ymin),fill=False,edgecolor='red', linewidth=2))
            #给方框加标注，xmin,ymin 表示 x,y 坐标，其它相当于画笔属性
            ax.text(xmin,ymin,s = str(num_class[classes[box_index]])+str(scores[box_index]),
                style='italic',bbox={'facecolor': 'blue', 'alpha': 0.5, 'pad': 0})
        plt.show()
```

步骤 6　测试结果输出

```
context.set_context(mode=context.GRAPH_MODE, device_target="Ascend")

ckpt_path = './ckpt/'
if not os.path.exists(ckpt_path):
    mox.file.copy_parallel(src_url=args_opt.ckpt_url, dst_url=ckpt_path)
cfg.ckpt_path = os.path.join(ckpt_path, "yolov3-60_15.ckpt")

data_path = '../data/'
if not os.path.exists(data_path):
    mox.file.copy_parallel(src_url=data_url, dst_url=data_path)

mindrecord_dir_test = os.path.join(data_path,'mindrecord/test')
prefix = "yolo.mindrecord"
cfg.mindrecord_file = os.path.join(mindrecord_dir_test, prefix)
cfg.image_dir = os.path.join(data_path, "test")
if os.path.exists(mindrecord_dir_test) and os.listdir(mindrecord_dir_test):
    print('The mindrecord file had exists!')
else:
    if not os.path.isdir(mindrecord_dir_test):
        os.makedirs(mindrecord_dir_test)
    prefix = "yolo.mindrecord"
    cfg.mindrecord_file = os.path.join(mindrecord_dir_test, prefix)
    print("Create Mindrecord.")
    data_to_mindrecord_byte_image(cfg.image_dir, mindrecord_dir_test, prefix, 1)
    print("Create Mindrecord Done, at {}".format(mindrecord_dir_test))
    # if you need use mindrecord file next time, you can save them to yours obs.
    #mox.file.copy_parallel(src_url=args_opt.mindrecord_dir_test,
dst_url=os.path.join(cfg.data_url,'mindspore/test')
print("Start Eval!")

yolo_eval(cfg)
```

输出（WARNING 不用管）：

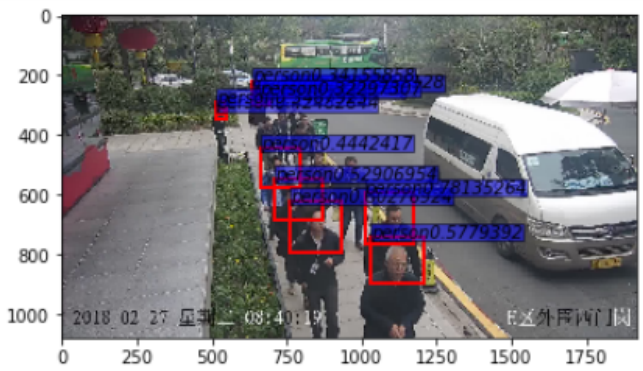
```
Create Mindrecord.
Create Mindrecord Done, at ./data/mindrecord/test
Start Eval!
```

```
Load Checkpoint!

=========================================

total images num:  8
Processing, please wait a moment.
```

图1.4.7.1.1.6.1.1 输出截图



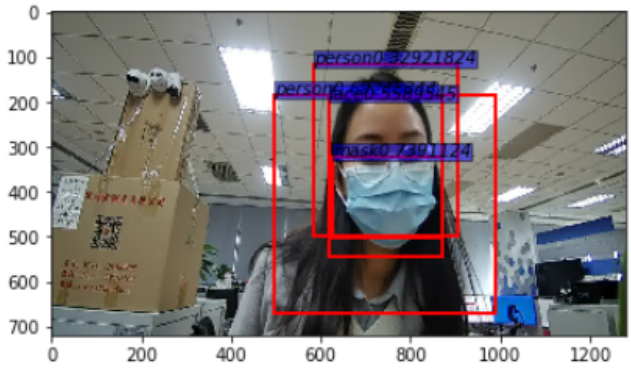


图1.4.7.1.1.6.1.2 测试图片效果

1.5 实验总结

本实验主要介绍如何使用 MindSpore 在利用 YOLOv3 网络模型实现目标检测任务。通过本实验学员将了解如何使用 MindSpore 深度学习框架实现 YOLOv3 目标检测网络模型的开发过程，通过基于该框架的训练和推理过程，进一步增加实践能力。